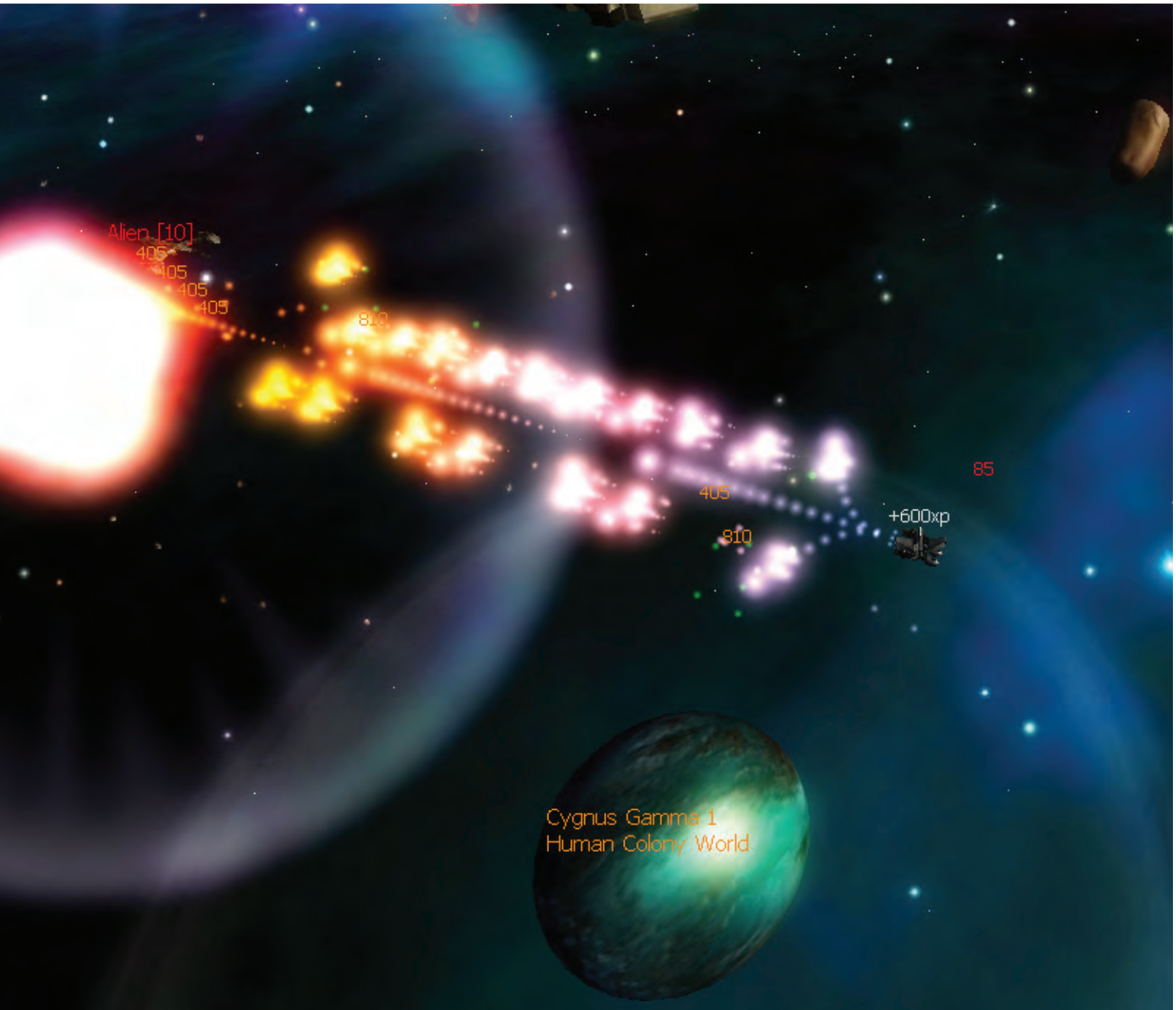
SIGEVOlution

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

Volume 4
Issue 4



in this issue

Editorial

And then there were four. This issue ends the fourth volume of SIGEVOlution and clears the backlog accumulated in 2009. The newsletter is now back on schedule! It seems like a long time since the first issue came out in May 2006. Since then, the newsletter published 33 articles, three interviews, several event reports and thesis summaries. The most downloaded issue on its publication day is the one hosting John Holland's interview, with more than 600 downloads.

As the cover suggests, the first article of this issue is about games, more precisely, about Galactic Arms Race (GAR), an indie video game developed by the Evolutionary Complexity Research Group (EPlex) at the University of Central Florida. GAR was selected as a finalist in the Indie Game Challenge and won the Editor's Pick for Best AI in an Independent Game in the 2009 AIGameDev.com Award. Next, Juan-Julián Merelo briefly presents his Perl library to help researchers working with evolutionary algorithms. At the end, the columns provide information about new issues of journals and forthcoming events.

This new issue comes to you thanks to Erin J. Hasting, Kenneth O. Stanley, Juan-Julián Merelo, Martin Butz, Xavier Llorà, Kumara Sastry, Cristiana Bolchini, Francesco Amigoni, Mario Verdicchio, Viola Schiaffonati, and board members Dave Davis and Martin Pelikan.

The cover is a screenshot from Galactic Arms Race.

Pier Luca
March 31st, 2010

SIGEVOlution Volume 4, Issue 4

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

Contributors to this Issue

Erin J. Hastings
Kenneth O. Stanley
Juan-Julián Merelo

Contents

ISSN: 1931-8499

Galactic Arms Race: An Experiment in Evolving Video Game Content

Erin J. Hastings, University of Central Florida, Orlando, hastings@cs.ucf.edu
Kenneth O. Stanley, University of Central Florida, Orlando, kstanley@cs.ucf.edu

Galactic Arms Race (GAR)[1] is an indie video game developed by the Evolutionary Complexity Research Group (EPlex) at the University of Central Florida to demonstrate the potential for novel artificial intelligence (AI) technology to impact video games. In particular, the new technology in GAR is an evolutionary algorithm called *content-generating neuroevolution of augmenting topologies* (cgNEAT), which is designed to evolve unique game content as the game is played. GAR is a multi-player space shooter in which players fight with particle-system weapons. The unique feature of GAR is that the game continually introduces new such weapons evolved by the cgNEAT algorithm. The philosophy behind GAR is that one of the best ways to support the argument that novel AI algorithms can impact how games are made is to make a fun game that is not only an academic proof-of-concept, but also a genuinely entertaining experience for regular gamers.

Because of this philosophy, the story behind GAR and its development is somewhat unique, although it follows in part from the earlier experience of co-developer Kenneth Stanley on the NERO project [8]. GAR began both as a dissertation project in AI by Erin Hastings and as a volunteer project for undergraduate and Masters students looking for experience in game programming. While significant effort centered on the content-generating algorithm, game design and architecture were also important focuses. That is, for the project to meet its goals, it had to be fun and attract players from outside the academic world. In fact, thorough quantitative analysis of the idea (which is necessary for it to be published) would require attracting a significant user-base just to test the algorithm, which is designed to leverage input from many simultaneous players.

Fortunately, the effort did ultimately pay off. GAR won the Best Paper Award at the 2009 IEEE Symposium on Computational Intelligence in Games [3], Editor's Pick for Best AI in an Independent Game in the 2009 AIGameDev.com Awards[2], and was chosen as one of 12 finalists out of over 250 entries to the 2010 Indie Game Challenge[3]. It appeared in Slashdot and a number of other media sources[4] and was downloaded over 10,000 times in the weeks after its release. In the GAR online multiplayer experiment [2], over 1,000 registered players from across the world evolved literally hundreds of thousands of weapons and destroyed over *one million* virtual aliens. Thus the project was able to accumulate a significant volume of data on the behavior of players and the outcomes of evolutionary content generation in a real-time online multiplayer game.

In principle, cgNEAT can evolve any class of parameterized content. Thus the generality of the approach means it may impact future commercial game production and increase the longevity of games that might otherwise become repetitive, potentially bringing a new application of evolutionary computation to industry. In this way, the results in this paper open up a promising new direction in video game design and research.

This article gives a brief overview of cgNEAT and GAR. A more detailed description of these techniques and related works can be found in Hastings, Guha, and Stanley [2].

[1] The GAR multiplayer demo is available at http://gar.eecs.ucf.edu/.

[2] http://aigamedev.com/open/editorial/2009-awards-results/

[3] http://www.indiegamechallenge.com/finalists/galactic-arms-race/

[4] http://gar.eecs.ucf.edu/index.php?content=media

Collaborative Content Evolution (CCE)

Evolving procedural game content is an emerging research area with great potential to contribute to the mainstream gaming industry. There are some examples of evolved game content that predate GAR, including race tracks [13] and even the rules of the game itself [14, 1].

Other recent work begins to bridge the gap between evolutionary art and games. In Avera [1, 4], the system evolves interactive art pieces for simple puzzle games. In another example [15], players interact with complex swarm systems through an IEC interface, enabling search for well-performing swarm configurations.

Inspired by these works, cgNEAT aims to evolve game content in real time, based on tracked player preferences in a multiplayer setting through a process called *collaborative content evolution* (CCE), as illustrated in figure 1.

In short, players begin the game with an initial set of content. If players use some of the content often, it is inferred that they enjoy that content, and the game produces new content that extends from or elaborates on that preferred content. However, if players are unhappy with certain content, they will not use it (or may discard it); thus the game will not produce more content of that type. In this manner, content is continually evolved based on the preferences of the players.

Content Generating NeuroEvolution Of Augmenting Topologies (cgNEAT)

The goal of the cgNEAT algorithm is to automatically generate computer graphics and video game content based on user behavior as the game is played. While there are technologies for evolving content like pictures [6, 12, 5], such technologies are not designed to work in real time during a game; rather they require users to explicitly designate which items are the best, which is something that a user playing a game does not want to do. That is, constantly answering questions about what they like and what should be produced in the future would disrupt players' experience. In contrast, the cgNEAT method makes these decisions automatically based on implicit information within usage statistics.

The main principles of cgNEAT follow:

1. Each content item is represented by a genome. The genome in GAR is a special kind of neural network called a *compositional pattern producing network* (CPPN) [7] that guides how particle weapons behave. In principle, a different representation than CPPNs can also be evolved.
2. During the game, each content item is assigned a fitness that is computed based on how often players actually *use* the content. That way, the system knows the relative popularity of each content item currently in the game.
3. Players begin the game with either (1) random content or (2) content from the *starter pool*, which is a special pool of content appropriate to beginners in the game. Starter pool content *does not* contribute to evolution and cannot be selected for reproduction.
4. Content is spawned in the game world, which means that it is placed in parts of the world where users can obtain it. However, unlike in most evolutionary systems, spawned content is not eligible for reproduction until players pick it up.
5. Content is reproduced in cgNEAT as follows: The algorithm selects content items from among content that players in the world *already possess* as parents that reproduce to form new content, which is spawned as described in step 4. The content items that are chosen as parents are selected probabilistically based on a roulette wheel scheme in which the chance of being chosen as a parent is proportional to the popularity (i.e. fitness) of the item. Reproduction, including mutation and crossover, is performed based in part on the NEAT algorithm [9, 10]. Thus, there is a chance that CPPNs may become more complex than their parents.
6. For any new content that is spawned, there is a probability (selected by the designer) that it will be chosen from a *spawning pool*, which is a collection of pre-evolved content, instead of being reproduced from parents. This pool ensures that diversity is not lost and that good types of content from the past (i.e. those that users liked) might reappear. Additionally, it ensures an initial seed of good content when the game first starts and players' preferences are unknown. The game designers initially select content, which may be pre-evolved before the game is released, to include in the spawning pool.

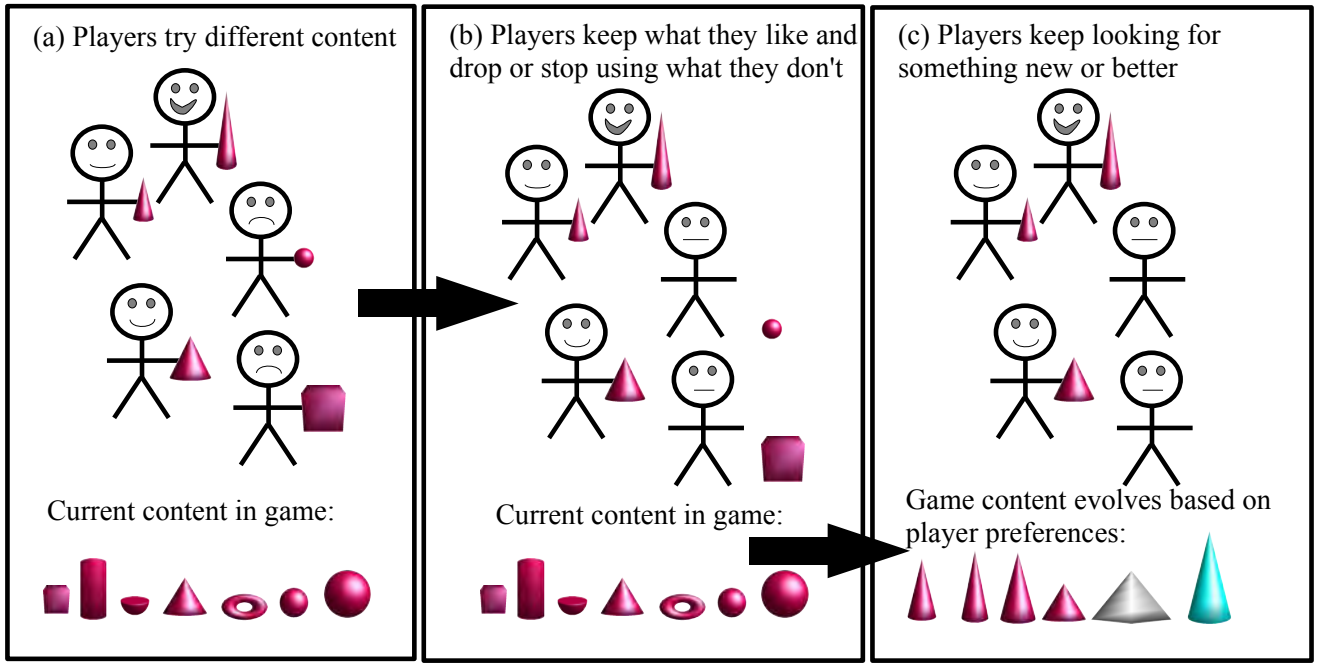


Fig. 1: **Illustrating Collaborative Content Evolution (CCE) in Games**. The main idea in CCE is that content evolution in games begins with a diverse population of randomized content (left). As players explore the world and discover new content, they likely keep content with which they are satisfied and discard that with which they are not (center). As evolution continues, content that is widely disliked filters out of the game (right) and content that players enjoy becomes the parents of new generations of content. In this way, players continually explore a succession of changing content.

7. Content that obtains a very high fitness (i.e. is popular with players) may optionally be saved to the *content archive*. There are several ways game designers can use archive material including (1) data analysis, (2) cycling it back into the game by adding it to the spawning pool, or (3) giving it to NPCs for use against players.

Finally, note that for cgNEAT in a multiplayer environment, although all players' preferences directly contribute to the course of evolution, the content generated is not an "average" of all player preferences. Rather, unique content is reproduced on a individual basis from individual items that are popular. Thus several diverse trends can flourish simultaneously.

Note that the cgNEAT algorithm incorporates some mechanics of its predecessor NEAT [9, 10] and standard evolutionary computation (EC), yet exhibits several major differences. Unlike in normal EC, the population size (i.e. those items that are eligible at any given time to reproduce) is variable and depends entirely on the number of users in the system. Furthermore, when an offspring is produced, unlike in normal evolutionary computation, it is not immediately placed into the population eligible to reproduce. Instead, it is in a special temporary state (placed somewhere in the game world) in which it may join the population only if a user chooses to acquire it. Also unlike normal evolutionary computation, instead of fitness determining which items are eliminated from the population, users entirely determine which items leave the population simply by discarding them.

Unlike standard interactive evolutionary computation (IEC [11]), users never explicitly communicate to the system which content they like. Instead, the preferred content is induced by the system implicitly from natural human behavior. That is, users do not need to know that they are interacting with an evolutionary algorithm yet evolution still works anyway.

Unlike regular NEAT, speciation is not necessary because users determine what is popular and the diversity of the population reflects the diversity of user preferences. Every step of the cgNEAT algorithm is asynchronous. At any time players may cause content to join the population or be eliminated.

Galactic Arms Race

The cgNEAT algorithm was introduced to evolve game content in real time, based on tracked player preferences in a multiplayer setting through CCE. This idea was first introduced in the Galactic Arms Race (GAR [2]) video game. In GAR (figure 2), the goal is to pilot a space ship to defeat enemies, gain experience, earn money, and most importantly, to find advantageous new weapons that are automatically generated by the cgNEAT algorithm [2]. GAR contains both a single player game and a full multiplayer game, in which weapons evolve based on the aggregate usage of all players online. In single player mode, evolution is directed by the actions of a single player battling NPC aliens in the game. GAR multiplayer evolution is substantially more diverse because the evolutionary population consists of the weapons currently possessed by all players in the game (i.e. it is CCE).

Each weapon in GAR is unique and represented by a CPPN [7], which is a type of artificial neural network (ANN). In GAR, players begin the game with an initial set of particle system weapons, which is the class of content evolved by the game. If players fire certain weapons often, cgNEAT infers that they enjoy that content, and the game produces new content that extends from or elaborates on that preferred content. In contrast, if players are unhappy with a certain weapon, they will not use it (or may discard it); thus the game will not produce more content of that type. The aim of this process is to continually evolve content based on the preferences of the players.

During the online multiplayer experiment in 2009, 1,007 unique player accounts were created on the server in approximately two months. At the time of the sample, over 73% of valid player accounts progressed past level 20, indicating substantial time (i.e. at least two or three hours) invested. A substantial proportion of players progressed to much higher levels, which takes several days in-game. The primary method of obtaining new weapons in GAR is to defeat enemies. The 1,007 players on the server scored 9,038 player-versus-player (PVP) kills, 721,456 Alien kills, 714,274 Pirate kills, and 22,670 Space Blob kills. Such a large kill total (over 1.46 *million*) hints at the intensity of game play.

Fig. 2: **GAR Client**. Players in GAR pilot their space ship (screen center) from a third-person perspective. This picture demonstrates a player destroying enemies with an evolved weapon. Left of the player ship is a weapon pickup dropped from a destroyed enemy base. A particle system preview emits from the weapon pickup (i.e. "neuralium isotope," left of player) to visually indicate how the weapon will function before the player picks it up. The GAR Client software is available online at http://gar.eecs.ucf.edu and runs on any Windows PC.

In total, 379,081 weapons were evolved (note that about 10% of these are from the spawning pool) by players destroying enemies, their bases, and other players. This number is remarkably high for an IEC system. On average, each player encountered over 375 weapon drops. Additionally, players fired over *23.6 million* shots with the evolved weapons they discovered. These results indicate that cgNEAT is capable of exposing players to a large variety of content quickly.

When weapons are spawned in the galaxy, they are displayed *before* they are picked up as a visualization called a "neuralium isotope." Thus players can decide whether or not to pick up a weapon based on its visualization. Of the 379,081 weapons dropped, 132,722 were picked up by players, which is roughly 35%. The reasons that players do not always pick up weapons are either (a) the weapon is deemed inferior to those in their arsenal, or (b) the weapon is similar to a weapon they already possess. In this context, that 35% of all weapons are picked suggests that players indeed decide whether or not to pick up content based on the previews. That is, players do not need to pick up every weapon they see. At the same time, it suggests that a considerable proportion of weapons evolved (about one third) are attractive enough to pick up, even with only three weapon slots available.

The starter weapons in GAR (which shoot in a straight line) act as an experimental control to compare with the weapons evolved by cgNEAT. During the snapshot, the number of starter weapons possessed by players above level 50 was 70, which is only 4% of the total weapons held by those players. Note that in GAR players are able to obtain starter weapons at any time during the game by selling unwanted isotopes in exchange for starter weapons. In fact, because such a sale also yields several credits of game currency for the player, in effect there is an incentive to sell evolved weapons in exchange for starter weapons. Nevertheless, of the over 1.46 million kills of players and NPCs on the GAR official server, only 22,935 were by starter weapons (roughly 1.5%). From these results it can be inferred through player behavior that they preferred evolved weapons to starter weapons.

The total number of combined *generations* of all weapon lineages in the snapshot is 50,646, and the highest generation weapon is 98, suggesting that weapons continue to be used even into later generations. Additionally, the average number of generations per weapon in the population sample is 16, indicating that it does not take many generations for players to find weapons that they want to keep.

Overall, players in the GAR multiplayer experiments discovered a wide variety of both aesthetically and tactically diverse weaponry evolved through their implicit preferences. Additional server data that was stored includes the *weapon archive*, where all weapons that were fired at least 800 times (i.e. highly fit weapons) are saved, and the *PVP archive*, where all weapons that score PVP kills are stored. By the time of the snapshot, these archives contained 5,209 and 1,662 weapons, respectively. The weapons presented in figure 3 from those archives were evolved by players on the official server from around the world.

Discussion and Future Work

The hope for GAR is that it shows that evolutionary computation may be a key enabling technology for automatic content generation. In the future, cgNEAT may evolve a different kind of content in an entirely different game. For example, vehicles, clothing, buildings, decorations, avatars, and other types of weapons all might evolve according to the usage-based heuristic in cgNEAT. A massive multiplayer online game (MMO) could be an ideal setting for such a process because the persistence of the world means evolution can continue for months or years.

The usage statistics, diversity of weapons, and recognition that GAR has received suggest that it is possible to succeed in this type of indie project in a low-budget academic setting. It also provides an opportunity for undergraduates to gain valuable experience working on a video game with a real game engine. For graduate students, the game can serve as a platform for Ph.D.-level research. The potential for wider recognition, which is greater in the game industry than in some other domains, also provides significant motivation to student teams.

In a larger context, GAR suggests a general model for research projects in AI and gaming. Instead of viewing existing games as testbeds for competing algorithms, an appealing alternative is to build an entirely new game around a technology to demonstrate its potential to deliver engaging new experiences. An additional benefit is that potential players need not purchase an existing game to participate, because the experimental game can be distributed freely on the Internet. Evolutionary computation can play a role in enabling the next generation of indie games, which are an effective medium for communicating the potential of evolution to the public as well as to fellow researchers.

(a) 3 gens (b) 12 gens (c) 11 gens

(d) 9 gens (e) 27 gens (f) 4 gens

Fig. 3: **Evolved Weapons**. This example displays archived weapons from the GAR Official 32-player server. Note that none of these weapons were conceived by the developers; all of them were evolved by cgNEAT based on the preferences of players.

Acknowledgments

Special thanks to the Galactic Arms Race (GAR) development team, testers, and everyone who has downloaded the game since its release. The Galactic Arms Race free demo is available online at http://gar.eecs.ucf.edu and the project's official email address is gar@eecs.ucf.edu.

Evolutionary Complexity Research Group at UCF

The Evolutionary Complexity Research Group at UCF (also known as EPlex; http://eplex.cs.ucf.edu) is directed by Prof. Kenneth Stanley. Its primary focus is on developing algorithms inspired by the ability of evolution in nature to produce highly complex artifacts. These algorithms are then applied to a variety of domains, including neuroevolution, automated control, multiagent learning, art, music, and video games. Algorithms and software developed by EPlex include HyperNEAT, multiagent HyperNEAT, novelty search, Picbreeder, NEAT Particles, and Galactic Arms Race. All are freely available from the EPlex website.

Bibliography

[1] S. Colton and C. Browne. Evolving simple art-based games. *Applications of Evolutionary Computing*, 5484, 2009.

[2] E. Hastings, R. Guha, and K. Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4), 2009.

[3] E. J. Hastings, R. K. Guha, and K. O. Stanley. Evolving content in the galactic arms race video game. *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2009.

[4] M.Hull and S. Colton. Towards a general framework for program generation in creative domains. *Proceedings of the 4th International Joint Workshop on Computational Creativity*, 2007.

[5] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley. Picbreeder: Evolving pictures collaboratively online. In *Proceedings of the Computer Human Interaction Conference*, 2008.

[6] K. Sims. Artificial evolution for computer graphics. *Proceedings of the ACM Special Interest Group on Graphics and Interactive Techniques*, pages 319–328, 1991.

[7] Kenneth O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, 8(2):131–162, 2007.

[8] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation Special Issue on Evolutionary Computation and Games*, 9(6):653–668, 2005.

[9] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[10] Kenneth O. Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004.

[11] H. Takagi. Interactive evolutionary computation: Fusion of the capacities of EC optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, 2001.

[12] S. Todd and W. Latham. *Evolutionary Art and Computers*. Academic Press, Inc., Orlando, FL, USA, 1992.

[13] J. Togelius, R. De Nardi, and S. M. Lucas. Towards automatic personalised content creation for racing games. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*. IEEE Press, 2007.

[14] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*. IEEE Press, 2008.

[15] S. von Mammen. Swarming for games: Immersion in complex systems. *Applications of Evolutionary Computing*, 5484, 2009.

About the authors

Kenneth O. Stanley is an assistant professor in the School of Electrical Engineering and Computer Science at the University of Central Florida. He received a B.S.E. from the University of Pennsylvania in 1997 and received a Ph.D. in 2004 from the University of Texas at Austin. He is an inventor of the Neuroevolution of Augmenting Topologies (NEAT) and HyperNEAT algorithms for evolving complex artificial neural networks. His main research contributions are in neuroevolution (i.e. evolving neural networks), generative and developmental systems, coevolution, machine learning for video games, and interactive evolution. He has won separate best paper awards for his work on NEAT, NERO, NEAT Drummer, HyperNEAT, novelty search, and Galactic Arms Race. He is the chair of the IEEE Task Force on Computational Intelligence and Video Games, associate editor for IEEE Transactions on Computational Intelligence and AI in Games, and is chairing the AI Video Competition at AAAI in 2010.

Homepage: www.cs.ucf.edu/~kstanley

Erin J. Hastings earned a Ph.D. in Computer Science at the University of Central Florida in 2009. He received a B.S. in Computer Science from University of Florida in 2001 and an M.S. in Computer Science from University of Central Florida in 2004. His research interests include procedural game content, IEC, particle systems, spatial subdivision, and networking. He has recently published papers in IEEE Transactions on Computational Intelligence and AI in Games and IEEE Transactions on Evolutionary Computation.

Homepage: www.eecs.ucf.edu/~hastings

Evolutionary Complexity Research Group at UCF: eplex.cs.ucf.edu

FOGA 2011 - Foundations of Genetic Algorithms

January 5-9, 2011, Schwarzenberg, Austria
http://www.sigevo.org/foga-2011
Enquiries and Submissions: foga@fhv.at
Deadline Monday July 5, 2010

We invite submissions of extended abstracts for the eleventh Foundations of Genetic Algorithms workshop. FOGA is only held every two years and focuses on theoretical foundations of all flavors of evolutionary computation. It will next be held in the Gasthof Hirschen hotel in Schwarzenberg in Austria from Wednesday, January 5 to Sunday January 9, 2011. Prof. Dr. Karl Sigmund has agreed to deliver a keynote lecture. Attendance is limited to people who submitted papers, or those requesting attendance in advance. Students are particularly encouraged to participate.

Submissions should address theoretical issues in evolutionary computation. Papers that consider foundational issues, place analysis in the wider context of theoretical computer science, or focus on bridging the gap between theory and practice are especially welcome. This does not prevent the acceptance of papers that use an experimental approach, but such work should be directed toward validation of suitable hypotheses concerning foundational matters.

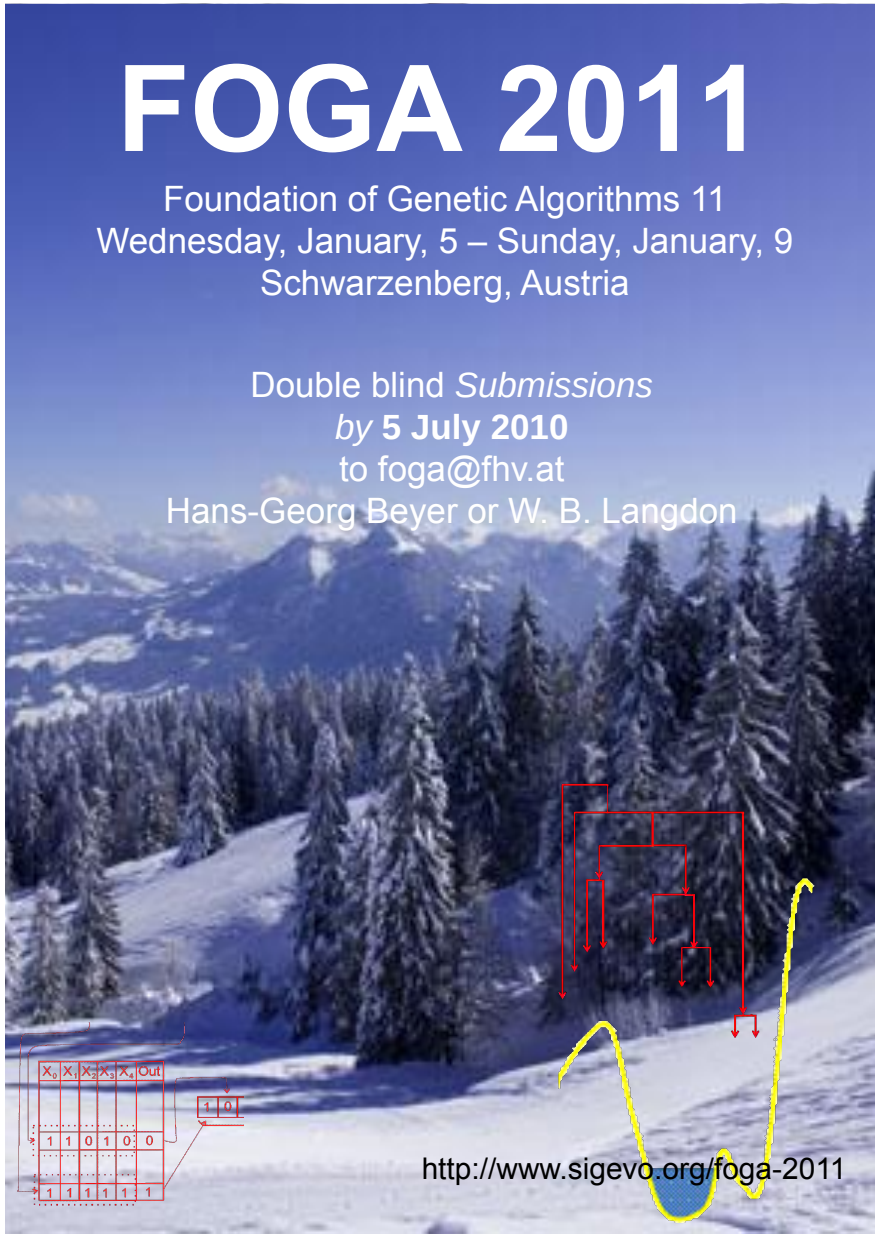
Extended abstracts should be between 10-12 pages long. To submit, please email a compressed postscript or a PDF file to foga@fhv.at no later than Monday, July 5, 2011. In your email, also include the title of the paper, and the name, address and affiliation of all the authors. To ensure the reviews are double-blind authors are asked to remove references to themselves from their paper.

Important Dates

Extended abstracts due	July 5, 2010
Notification to authors	September 13, 2010
Registration and room booking deadline	October 8, 2010
Pre-proceedings camera ready manuscript due	December 6, 2010
FOGA workshop	January 5–9, 2011
Post workshop proceedings	February 21, 2011

Organizers

Prof. Dr. habil. Hans-Georg Beyer	www2.staff.fh-vorarlberg.ac.at/ hgb/
Dr. W. B. Langdon	www.dcs.kcl.ac.uk/staff/W.Langdon/



A Perl Primer for Evolutionary Algorithm Practitioners

Juan-Julián Merelo, Dept. Arquitectura y Tecnologìa de Computadores, University of Granada, Spain, jj@merelo.net

It is curious that the Perl language is not widely used within the Evolutionary Algorithms community, since the main virtues of a Perl programmer can also be applied to the general EA practitioner. These virtues are Impatience, Laziness and Hubris [3], and it is evident that we try to evolve something from the initial, random, chaos, appearing then as godlike creatures to our humble chromosomes with high fitness, thus being prone to the most important of the triad, Hubris. And who's not to say that evolutionary algorithms are for lazy slobs? Instead of devising an analytical solution to a problem, or greedily building it piece by painstaking and carefully designed piece, we just cobble together a fitness function and let the evolutionary algorithm have a go at the problem. If anything, we might fiddle a bit with operator rates, but not much more; we do not even care about understanding the problem (we leave that for memetic algorithm practitioners). Just the fitness, ma'am, no need to do any hard work.

And nobody can deny we are impatient, trying to make faster what is already fast, and throwing 256-node clusters at teeny problems to find the solution in no time flat. Instead of letting stuff evolve at the leisurely million-year pace of real evolution, we throw iron to it to make it fast, and faster, and instead of doing a sure-fire exhaustive search, which is bound to find the solution (sooner or later, in this present universe or the next one), alas, we try to find it without even looking into the last nook and cranny of search space.

So, having so much in common, I think it would be interesting to do evolutionary algorithms in Perl. And since you (and I) are impatient persons, let's jump straight to it, and not waste our time in disquisitions.

Chromosomes and Populations

Let's start by doing something simple: mutate a chromosome. We are bound to do it sooner or later, so the sooner the better. The program shown in Table 1 would do it.

```
#!/usr/bin/perl

my $chromosome = shift || "11111111";

my $mutation_point = rand( length( $chromosome ));
substr($chromosome, $mutation_point, 1,
       ( substr($chromosome, $mutation_point, 1) eq   1 )?0:1
);
print $chromosome, "\n";
```

Tab. 1: Single chromosome mutation.

In fact, it would do a bit more than that: it would take a string issued as a command line parameter (with a default value of `11111111`), generate a random point within it, and flip its value. The first line is probably the weirder, since there is no mention of the command line anywhere. This is due to Perl's impatience: we want to do stuff as fast as possible, so that line actually means what we see in Table 2, which, in turn, is also a short version of the code shown in Table 3.

```
my $chromosome = shift @ARGV || "11111111";
```

Tab. 2: Equivalence of $foo = shift.

```
my $chromosome;
if ( $ARGV[0] ) {
  $chromosome = shift @ARGV;
} else {
  $chromosome = "11111111";
}
```

Tab. 3: Longest form of command-line processing.

What does it all mean? Declare a variable called `$chromosome`, which can hold anything as long as it is an scalar variable (number, boolean, string), check if there is anything on the command line (contained on the array ARGV, which is included in every program, and need not be declared; arrays, unlike scalars, use @ instead of $); if there is anything there, pluck the first element from the array and put it into the variable; if there is not, just declare the default value. In fact, the first version says the same, `||` reads "or" and acts as an alternative operator: if the right hand side does is null (it has not been able to extract anything form the command line parameters) then use the left hand side.

The rest is quite similar to other languages like C or Java except for the funky variable syntax. In fact, we need not declare them, only it's convenient. All `my`s could happily have been suppresed, and the program would still work.

We still need to run the program. Perl is an interpreted language, so running implies saving the program and either making the file runnable (via `chmod` in Linux) or running it this way:

```
prompt$ perl mutate.pl 111000111000
111000101000
```

You can do it again and again, until you find the solution. But since you are an EA practitioner (and now a Perl programmer too) you are too impatient, and also lazy, so it's better to create another program that does this for you, like the one shown in Table 4.

However, this is roughly an evolutionary algorithm from the fifties, where mutation is the only way to explore space and, thus, does not have a very good result. It does, however, show a few of the unique Perl features. Let us look, for instance, at the subroutine named `fitness`, by the end of the program. It is a simple conversion from binary to decimal, by appending "0b" at the beginning of a string, and forcing evaluation (via `eval`) to a number. Strings and numbers are interpreted contextually in Perl; an alphanumeric string will behave like one or the other depending on what we do with them; in this case we want it to be a number, that is why we evaluate. However, `eval`, as it happens in many other interpreted languages, is a function that allows you to build, on the fly, Perl structures and interpret them (is anyone hearing Genetic Programming here?).

The next-to-last subroutine is a subroutinization of the mutation function we saw before; it behaves pretty much as a whole program, receving its arguments by `shift`ing the argument array, which in this case is `@_`, which of course is not shown. We might find another interesting feature in the previous subroutine, the expression `1..$length`, which is nothing less than a list created from the two extremes. And once again, the loop variable is implicit, where the heck is it? Actually, nowhere, since we are not using it, but if we needed its value, we could access it using `$_`, the quintessential default variable in Perl. Whenever you are looking for a loop variable and it's not there, just try `$_`.

That is enough with the subroutines, let us look at the main program. After the usual *declaration* of variables (which is rather a scope declaration, since we do not need to declare them), comes one of the nice list handling functions of Perl: `map`, which does exactly that: map from one array to another, applying a function to every single element of the array referred to by the default variable `$_`. In that single statement, we create a list with all numbers from 1 to the number of elements on the population, and generate a random chromosome for eacn one. The actual number is not used in this case, but it is a couple of lines down the program: once the population is created, we evaluate it and put it in a hash (an array keyed by strings, instead of natural numbers).

This is incredibly useful: we will only need to evaluate a chromosome a single time, keeping the value for later use; this hash is keyed by chromosome, so it's stored a single time. In fact, this nice feature could be used much better, by changing the `fitness` subroutine to the one shown in Table 5, which guarantees that every member of the population is evaluated only once at the cost of using more memory, so be careful with this in limited memory environments.

```
my $chromosome_length = shift || 16;
my $population_size = shift || 32;
my $generations = shift || 10;

my @population = map( random_chromosome( $chromosome_length ),
        1..$population_size );

my %evaluated_population;
map( $evaluated_population{$_} = fitness( $_ ), @population );
for ( 1..$generations ) {
  my @sorted_population = sort { $evaluated_population{$b}
       <=> $evaluated_population{$a} } @population;
  pop @sorted_population;
  @population = @sorted_population;
  push @population, random_chromosome( $chromosome_length );
  $evaluated_population{ $population[$#population] } =
    fitness( $population[$#population] );
  print "Best $population[0], ", $evaluated_population{$population[0]}, "\n";
}

sub random_chromosome {
  my $length = shift;
  my $string = '';
  for (1..$length) {
    $string .= (rand >0.5)?1:0;
  }
  $string;
}

sub mutate {
  my $chromosome = shift;
  my $mutation_point = rand( length( $chromosome ));
  substr($chromosome, $mutation_point, 1,
  ( substr($chromosome, $mutation_point, 1) eq 1 )?0:1 );
  return $chromosome;
}

sub fitness {
  my $chromosome = shift;
  return eval "0b$chromosome";
}
```

Tab. 4: Naïve mutation based proto-genetic algorithm.

```
sub fitness {
  my $chromosome = shift;
  if ( $evaluated_population{$chromosome} ) {
    return $evaluated_population{$chromosome};
  } else {
    $evaluated_population{$chromosome} = "0b$chromosome";
  }
  return $evaluated_population{$chromosome};
}
```

Tab. 5: Alternative fitness function that uses a cache.

Whatever its form, the main algorithm loop sorts the population (using the fitness stored in the hash), then eliminates the last member of the population by popping it, and generates a random one, pushing it into the population. The weirdest part of this is probably how sorting is done; first thing to take into account is that both sides of the assignment are arrays; sort takes an expression and compares members of the array to be sorted by using it; the two members to be compared become `$a` and `$b` within that expression, and the operator `<=>`, which returns -1, 0 or 1 depending on the relative values.

You can even run the program using other values:

```
prompt$ perl population-mutate.pl 16 64 1000
Best 1111111001110011, 65139
Best 1111111001110011, 65139
```

but unless you give it some time, you'll arrive nowhere fast. So it's time for the heavy artillery: using crossover, and a better way of selecting individuals.

Going All The Way: Using Selection and Crossover

A full genetic algorithm, in fact, takes 100 lines in Perl; only what is new is shown in Table 6. This program, which can be downloaded (along the rest of the programs) from http://sl.ugr.es/perl-primer, includes a few dirty tricks from the Perl bag. The fitness function has been changed to compute the number of ones (instead of the binary value) thus:

```
my $copy = $chromosome;
    $fitness_of{$chromosome} = ($copy =~ tr/1/0/);
```

`tr` is an operator that substitutes, and returns the number of substitution. Here we do so; substitute the ones for 0, which is quite fast, and use the side effect as a result. Crossover is no big deal: uses `substr`, same as mutation, but two subroutines have been included for the stochastic universal sampling: `compute_wheel` and `spin`. The only new thing about these subroutines is that arrays are passed by reference (`\@array` is a reference to an array) and then de-referenced inside the subroutine (`$@array`).

Other than this sampling and crossovering, the program looks remarkably similar to the old one: sort population, extract the best from it, once again using array assignment: `@sorted_population[0,1]` are the first (index 0) and second element of the array, returned as an array too. We use also automatic conversion of arrays to scalars: an array like `@slots` becomes its number of elements in an scalar context such as

```
$p = $index++ % @slots;
```

In this case, this evolutionary algorithm behaves as it should: finds the solution quite fast, in just a few generations. And in 100 lines.

But in Perl, there is always another way to do it.

Using Algorithm::Evolutionary

Probably the single best thing about Perl is the existence of CPAN, the Comprehensive Perl Archive Network, which at http://cpan.org contains thousands of modules devoted to anything imaginable, and in fact several dozens of things nobody would be able to imagine (just look at the `Acme` modules). Needless to say, a search of *genetic* or *evolutionary* returns a few modules, but I will of course talk about my own, `Algorithm::Evolutionary` [2], which is not better than all the rest, but is probably bigger than any (and older, if that can be called a feature).

I have been using `Algorithm::Evolutionary` for all my evolutionary algorithm needs, the basic idea being that it should be easy for me to set up an experiment, repeat it over and over, and process results; the modules themselves are surrounded by a set of tools which are very useful in that sense. And this, in turn, has allowed me to publish every program used in my papers, along with the parameter files used to run them; you can reproduce any result I have obtained, and improve on it if you want.

In any case, using this standard (available from CPAN) library along with the files used to run everything makes all our experiments *reproducible*. In fact, files are available on the CVS server as I run them, although the CPAN releases are made usually after the paper is submitted (for lack of time, mainly).

That's enough for a spiel, let us see how this library improves what we have seen above. Is it easy to run a canonical genetic algorithm, for instance? See Table 7, which is part of the program included in the distribution, and is also available from `http://sl.ugr.es/cga_pl`.

The basic idea of this module is to provide a series of classes that can be instantiated to the most common objects in an evolutionary algorithm. The `Creator` module, for instance, is a type of class known as Factory [1] which takes as arguments the type of individuals we want to generate (`BitString`s), parameters they need, and population size, and, when applied, fills the population with objects of that class.

The program then proceeds to create the rest of the operators needed for the canonical GA: a `Bitflip` mutation, the usual 2-point `Crossover`; `Royal_Road` is used as fitness function, but in principle you can use anyone you want, programmed in Perl or (as a external program) in any other language. Then the `CanonicalGA` object is instantiated with everything we have created before: a fitness function, a selection rate, and a reference to an array (the square brackets) containing mutation and crossover.

Evolution is then a matter of applying this single-generation canonical GA repeatedly until a final number of generations is reached, or the solution is found (the fitness of the best is equal to the number of bits).

If you want to test this with your own fitness function, the only thing you have to change is exactly that; if you want to try with different parameters for mutation or crossover, those are changed from the command line. Finally, if you want to do your own operator, it is not so difficult either. It should not be, since, as a Perl programer/AE practitioner, you are... well, you know that already.

Do you want performance? Here's performance for you

It is quite usual to think that interpreted languages in general, and Perl in particular, are not suitable for algorithms whose implementations require high performance, such as evolutionary algorithms. Either in the purely evolutionary (which is essentially string processing, that is, integer processing) or the fitness computation area, compiled languages (C++ or Java) should theoretically be able to deliver more evaluations per second than anything written in Python, Javascript or Ruby. In fact, some general purpose benchmarks mention two orders of magnitude of difference between these types of languages.

It is impossible to discuss those claims, but AE practitioners are already used to know that There Is No Free Lunch. First, speed comes at the expense of less runtime flexibility, and more time to create a program. Laziness implies speed when sitting down to write a program and being able to obtain fast results for the paper you were supposed to submit yesterday. And second, some of these languages are optimized for certain tasks; Perl is quite fast when processing strings, and if you do a bit of experimenting and process them in the Perl way, you might be able to obtain better performance than if done in Java or machine code. And it is way faster than Matlab.

In fact, how fast is fast? Fast enough is good enough for me. While for certain fitness functions (for most, in fact) Perl will be slower than compiled C++, I value the huge amount of knowledge already in CPAN, which I can tap easily to build very complex fitness function, or, once again, which I can use to process experiment results, represent them graphically, or even interface with many other things: the R statistics package, a database, the web or anything else.

Let me finish this with a request: while nobody will tell you that you need to learn Perl to practice evolutionary algorithms, give Perl a try and you will discover that many things that you considered hard to do will become much easier. And your performance solving problems and writing papers about them will no doubt increase, which, as an impatient person, is probably what you are looking for.

Bibliography

[1] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing. Addison-Wesley.

[2] Merelo-Guervós, J.-J., Castillo, P.-A., and Alba, E. (2009). `Algorithm::Evolutionary`, a flexible Perl module for evolutionary computation. *Soft Computing*. To be published, accesible at `http://sl.ugr.es/000K`.

[3] Wiki (2009). Laziness impatience hubris. C2 wiki http://c2.com/cgi/wiki?LazinessImpatienceHubris.

About the author

Juan-Julián Merelo has been using Perl about as long as he's been into evolutionary computation, so it is only natural they would come together sooner or later. He is also full professor at the University of Granada, which makes him worry about the receding hairline that goes with it.

Homepage: http://geneura.ugr.es/~jmerelo/
Email: jj@merelo.net

```
my %fitness_of;
map( $fitness_of{$_} = fitness( $_ ), @population );
for ( 1..$generations ) {
  my @sorted_population = sort { $fitness_of{$b}
       <=> $fitness_of{$a} } @population;
  my @best = @sorted_population[0,1];
  print $best[0], " ", $fitness_of{$best[0]}, "\n";
  my @wheel = compute_wheel( \@sorted_population );
  my @slots = spin( \@wheel, $population_size );
  my @pool;
  my $index = 0;
  do {
    my $p = $index++ % @slots;
    my $copies = $slots[$p];
    for (1..$copies) {
      push @pool, $sorted_population[$p];
    }
  } while ( @pool <= $population_size );
  @population = ();
  map( mutate($_), @pool );
  for ( my $i = 0; $i < $population_size/2 -1 ; $i++ )  {
    my $first = $pool[rand($#pool)];
    my $second = $pool[ rand($#pool)];
    push @population, crossover( $first, $second );
  }
  map( fitness( $_ ), @population );
  push @population, @best;
}

sub compute_wheel {
  my $population = shift;
  my $total_fitness;
  map( $total_fitness += $fitness_of{$_}, @$population );
  my @wheel =
     map( $fitness_of{$_}/$total_fitness, @$population);
  return @wheel;
}
```

```
sub spin {
  my ( $wheel, $slots ) = @_;
  my @slots = map( $_*$slots, @$wheel );
  return @slots;
}

sub crossover {
  my ($chromosome_1, $chromosome_2) = @_;
  my $length = length( $chromosome_1 );
  my $xover_point_1 = int rand( $length -1 );
  my $xover_point_2 = int rand( $length -1 );
  if ( $xover_point_2 < $xover_point_1 )  {
    my $swap = $xover_point_1;
    $xover_point_1 = $xover_point_2;
    $xover_point_2 = $swap;
  }
  $xover_point_2 = $xover_point_1 + 1
     if ( $xover_point_2 == $xover_point_1 );
  my $swap_chrom = $chromosome_1;
  substr($chromosome_1, $xover_point_1,
     $xover_point_2 - $xover_point_1 + 1,
     substr($chromosome_2, $xover_point_1,
       $xover_point_2 - $xover_point_1 + 1) );
  substr($chromosome_2, $xover_point_1,
     $xover_point_2 - $xover_point_1 + 1,
     substr($swap_chrom, $xover_point_1,
        $xover_point_2 - $xover_point_1 + 1) );
  return ( $chromosome_1, $chromosome_2 );
}
```

Tab. 6: Genetic algorithm with fitness-proportional selection and 2-elitism. Common parts with the previous program have been suppressed for clarity.

```
my @pop;
my              $creator = new Algorithm::Evolutionary::Op::Creator( $pop_size,
'BitString', { length => $bits });
$creator->apply( \@pop ); #Generates population

my $m = Algorithm::Evolutionary::Op::Bitflip->new( 1 );
my $c = Algorithm::Evolutionary::Op::Crossover->new(2, 4);

my $rr = new  Algorithm::Evolutionary::Fitness::Royal_Road( $block_size );
map( $_->evaluate( $rr ), @pop );
my          $generation = Algorithm::Evolutionary::Op::CanonicalGA->new( $rr ,
$selection_rate , [$m, $c] ) ;

my $contador=0;
do {
  $generation->apply( \@pop );
  print "$contador : ", $pop[0]->asString(), "\n" ;
  $contador++;
} while( ($contador < $numGens)
  && ($pop[0]->Fitness() < $bits));
print "Best is:\n\t ",$pop[0]->asString()," Fitness: ",$pop[0]->Fitness(),"\n";
```

Tab. 7: Main part of a canonical genetic algorithm included with the `Algorithm::Evolutionary` Perl module.

New Issues of Journals

Evolutionary Computation 18(1) (www)

- **Analysis of an Asymmetric Mutation Operator**, Thomas Jansen, Dirk Sudholt, pp 1–26 (pdf)
- **Memetic Algorithms for Continuous Optimisation Based on Local Search Chains**, Daniel Molina, Manuel Lozano, Carlos García-Martínez, Francisco Herrera, pp 27–63 (pdf)
- **Computing Gap Free Pareto Front Approximations with Stochastic Search Algorithms**, Oliver Schütze, Marco Laumanns, Emilia Tantar, Carlos A. Coello Coello, El-Ghazali Talbi, pp 65—96 (pdf)
- **Adaptive Niche Radii and Niche Shapes Approaches for Niching with the CMA-ES**, Ofer M. Shir, Michael Emmerich, Thomas Bäck, pp 97—126 (pdf)
- **Strength Pareto Particle Swarm Optimization and Hybrid EA-PSO for Multi-Objective Optimization**, Ahmed Elhossini, Shawki Areibi, Robert Dony, pp 127—156 (pdf)

Evolutionary Intelligence 3(1) (www)

- **LSGA: combining level-sets and genetic algorithms for segmentation**, Payel Ghosh, Melanie Mitchell and Judith Gold, pp 1–11 (pdf)
- **Collective neuro-evolution for evolving specialized sensor resolutions in a multi-rover task**, G. S. Nitschke, M. C. Schut and A. E. Eiben, pp 13–29 (pdf)
- **A learning classifier system with mutual-information-based fitness**, Robert Elliott Smith, Max Kun Jiang, Jaume Bacardit, Michael Stout, Natalio Krasnogor and Jonathan D. Hirst, pp 31–50 (pdf)

Calls and Calendar

March 2010

4th Workshop on Theory of Randomized Search Heuristics (ThRaSH'2010)

March 24-25, 2010, Paris, France

Homepage: http://trsh2010.gforge.inria.fr/

Registration deadline: March 5, 2010

Following the workshops in Wroclaw, Poland, Dortmund, Germany, and Birmingham, UK, the 4th workshop on Theory of Randomized Search Heuristics (ThRaSH'2010) will take place in Paris on the 24th and 25th of March 2010. The purpose of the workshop is to address questions related to theory of randomized search heuristics such as evolutionary algorithms, ant colony optimization, or simulated annealing for both combinatorial and numerical optimization. The primary focus lies on discussing recent ideas and detecting challenging topics for future work, rather than on the presentation of final results.

Researchers working on theoretical aspects of randomized search heuristics are invited to submit a short abstract (one single page) by email to "thrash2010@lri.fr". No registration fee will be charged but participants are asked to register before the workshop.

April 2010

Evostar 2010 - EuroGP, EvoCOP, EvoBIO and EvoWorkshops

April 7-9, 2010, Istanbul Technical University, Istanbul, Turkey

Homepage: www.evostar.org

The EuroGP, EvoCOP, EvoBIO and EvoApplications conferences compose EVO*: Europe's premier co-located events in the field of Evolutionary Computing.

Featuring the latest in theoretical and applied research, EVO* topics include recent genetic programming challenges, evolutionary and other meta-heuristic approaches for combinatorial optimisation, evolutionary algorithms, machine learning and data mining techniques in the biosciences, in numerical optimisation, in music and art domains, in image analysis and signal processing, in hardware optimisation and in a wide range of applications to scientific, industrial, financial and other real-world problems.

EVO* Poster

You can download the EVO* poster advertisement in PDF format here (Image: Pelegrina Galathea, by Stayko Chalakov (2009))

EVO* Call for Papers

You can download the EVO* CfP in PDF format here.

EuroGP

13th European Conference on Genetic Programming

EvoCOP

10th European Conference on Evolutionary Computation in Combinatorial Optimisation

EvoBIO

8th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics

EvoApplications 2010

European Conference on the Applications of Evolutionary Computation

- EvoCOMNET: 7th European Event on the Application of Nature-inspired Techniques for Telecommunication Networks and other Parallel and Distributed Systems
- EvoCOMPLEX (new): Evolutionary Algorithms and Complex Systems
- EvoENVIRONMENT: Nature Inspired Methods for Environmental Issues
- EvoFIN: 4th European Event on Evolutionary and Natural Computation in Finance and Economics
- EvoGAMES: 2nd European event on Bio-inspired Algorithms in Games
- EvoIASP: EC in Image Analysis and Signal Processing
- EvoINTELLIGENCE: Nature Inspired Methods for Intelligent Systems
- EvoMUSART: 8th European event on Evolutionary and Biologically Inspired Music, Sound, Art and Design
- EvoNUM: 3rd European event on Bio-inspired algorithms for continuous parameter optimisation
- EvoSTOC: 7th European event on Evolutionary Algorithms in Stochastic and Dynamic Environments
- EvoTRANSLOG: 4th European Event on Evolutionary Computation in Transportation and Logistics

EvoPHD

5th European Graduate Student Workshop on Evolutionary Computation

Evo* Coordinator: Jennifer Willies, Napier University, United Kingdom
j.willies@napier.ac.uk

Local Chair: Şima Uyar, Istanbul Technical University, Turkey
etaner@itu.edu.tr

Publicity Chair: Stephen Dignum, University of Essex, United Kingdom
sandig@essex.ac.uk

July 2010

GECCO 2010 - Genetic and Evolutionary Computation Conference
July 7-10, 2010, Portland, Oregon, USA
Homepage: http://www.sigevo.org/gecco-2010
Workshop Deadline March 25, 2010
Late Breaking Papers Deadline April 13, 2010
Author notification: March 10, 2010
Camera-ready: April 5, 2010

The Genetic and Evolutionary Computation Conference (GECCO-2010) 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.

Organizers

General Chair:	Martin Pelikan
Editor-in-Chief:	Jürgen Branke
Local Chair:	Kumara Sastry
Publicity Chair:	Pier Luca Lanzi
Tutorials Chair:	Una-May O'Reilly
Workshops Chair:	Jaume Bacardit
Competitions Chairs:	Christian Gagné
Late Breaking Papers Chair:	Daniel Tauritz
Graduate Student Workshop	Riccardo Poli
Business Committee:	Erik Goodman
	Una-May O'Reilly
EC in Practice Chairs:	Jörn Mehnen
	Thomas Bartz-Beielstein,
	David Davis

Important Dates

Paper Submission Deadline	January 13, 2010
Decision Notification	March 10, 2010
Camera-ready Submission	April 5, 2010

Venue

The Portland Marriott Downtown Waterfront Hotel, located in downtown Portland, is near the Portland Riverplace Marina, restaurants, shopping & performing arts venues. Hotel room conference rate $179 includes complimentary in-room high-speed Internet access.

More Information

Visit www.sigevo.org/gecco-2010 for information about electronic submission procedures, formatting details, student travel grants, the latest list of tutorials and workshop, late-breaking papers, and more.

For technical matters, contact Conference Chair Martin Pelikan at pelikan@cs.umsl.edu.
For conference administration matters contact Primary Support Staff at gecco-admin@tigerscience.com.

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

2010 IEEE World Congress on Computational Intelligence
July 18-23, 2010, Barcelona, Spain
Homepage: WWW

The 2010 IEEE World Congress on Computational Intelligence (IEEE WCCI 2010) is the largest technical event in the field of computational intelligence. It will host three conferences: the 2010 International Joint Conference on Neural Networks (IJCNN 2010), the 2010 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2010), and the 2010 IEEE Congress on Evolutionary Computation (IEEE CEC 2010). IEEE WCCI 2010 will be held in Barcelona, a Mediterranean city located in a privileged position on the northeastern coast of Spain. Barcelona combines history, art, architecture, and charm within a pleasant, and efficient urban environment where meet old friends, and make new ones. The congress will provide a stimulating forum for scientists, engineers, educators, and students from all over the world to discuss and present their research findings on computational intelligence.

Important Due Dates

- Submission deadline: January 31, 2010
- Notification of paper acceptance: March 15, 2010
- Camera ready submission: May 2, 2010
- IEEE WCCI 2010 Conference: July 18-23, 2010

For more information visit http://www.wcci2010.org/call-for-papers

August 2010

IEEE Conference on Computational Intelligence and Games (CIG-2010)

August 18-21, 2010, Copenhagen, Denmark

Homepage: http://game.itu.dk/cig2010

Submission deadline: March 15, 2010

Decision notification: May 15, 2010

Camera-ready submission: June 15, 2010

Conference: August 18-21, 2010

Aim and Scope

Games have proven to be an ideal domain for the study of computational intelligence as not only are they fun to play and interesting to observe, but they provide competitive and dynamic environments that model many real-world problems. Additionally, methods from computational intelligence promise to have a big impact on game technology and development, assisting designers and developers and enabling new types of computer games. The 2010 IEEE Conference on Computational Intelligence and Games brings together leading researchers and practitioners from academia and industry to discuss recent advances and explore future directions in this quickly moving field.

Topics of interest include, but are not limited to:

- Learning in games
- Coevolution in games
- Neural-based approaches for games
- Fuzzy-based approaches for games
- Player/Opponent modeling in games
- CI/AI-based game design
- Multi-agent and multi-strategy learning
- Applications of game theory
- CI for Player Affective Modeling
- Intelligent Interactive Narrative
- Imperfect information and non-deterministic games
- Player satisfaction and experience in games
- Theoretical or empirical analysis of CI techniques for games
- Comparative studies and game-based benchmarking
- Computational and artificial intelligence in:
  - Video games
  - Board and card games
  - Economic or mathematical games
  - Serious games
  - Augmented and mixed-reality games
  - Games for mobile platforms

The conference will consist of a single track of oral presentations, tutorial and workshop/special sessions, and live competitions. The proceedings will be placed in IEEE Xplore, and made freely available on the conference website after the conference.

Conference Committee

General Chairs:	Georgios N. Yannakakis and Julian Togelius
Program Chair:	Michael Mateas, Risto Miikkulainen, and Michael Young
Proceedings Chair:	Pier Luca Lanzi
Competition Chair:	Simon Lucas
Local Chairs:	Anders Drachen, Paolo Burelli, & Tobias Mahlmann

Important Dates

Tutorial proposals:	31st January 2010
Paper submission:	15th March 2010
Decision Notification:	15th May 2010
Camera-ready:	15th Jun 2010
Conference:	18-21 August 2010

For more information please visit: http://game.itu.dk/cig2010/

September 2010

SSBSE 2010 - 2nd International Symposium on Search Based Software Engineering
September 7-9, 2010, Benevento, Italy
Homepage: www.ssbse.org
Deadline: April 23, 2010
Decision notification: 21 June 2010

We are pleased to announce SSBSE 2010, the second edition of the annual symposium dedicated to Search Based Software Engineering (SBSE). The symposium's objective is to build on the recent flourishing of interest in SBSE by not only creating a welcoming forum for discussion and dissemination, but also by establishing a regular event that will strengthen the rapidly growing international community.

Call for Papers

We invite the submission of high quality papers describing original and significant work in all aspects of Search Based Software Engineering, including theoretical work, research on SBSE applications, empirical studies, and reports from industrial experience. Applications may be drawn from throughout the software engineering lifecycle. Search methods may include, but are not limited to, operational research techniques and optimization methods inspired by nature, such as evolutionary algorithms and simulated annealing. We particularly encourage papers that use novel search techniques and describe software engineering applications to which SBSE has not previously been applied.

We also invite papers for a special PhD student track that will provide a venue for students to present their work and receive feedback from senior members of the SBSE research community. PhD papers may include thesis plans, proposed research, as well as partial or complete SBSE research results already obtained.

Fast abstracts may also be submitted that feature novel ideas, not yet fully developed or validated. Accepted fast abstracts will be presented at the symposium and published online on the conference website.

The authors of selected symposium papers will be invited to submit extended versions for a special issue of Information and Software Technology journal (IST) edited by Elsevier.

Keynote Speakers

- Prof. Riccardo Poli - University of Essex, UK
- Dr. Paolo Tonella - FBK-Irst, Trento, Italy

Organizing Committee

- General Chairs: Massimiliano Di Penta, Simon Poulding
- Program Chairs: Lionel C. Briand, John Clark
- PhD Forum Chair: Phil McMinn
- Fast Abstracts Chair: Gerardo Canfora
- Submissions Chair: Andrea Arcuri
- Web & Publicity Chair: Jan Staunton

Important Dates

- Technical papers/PhD forum submission: 23 April 2010
- Notification: 21 June 2010
- Fast abstract submission: 16 July 2010
- Notification: 23 July 2010

PPSN 2010 – International Conference on Parallel Problem Solving From Nature
September 11-15, 2010, Cracow, Poland
Homepage: http://home.agh.edu.pl/ppsn
Deadline: April 6, 2010

The Eleventh International Conference on Parallel Problem Solving from Nature (PPSN XI) will be held at the AGH University of Science and Technology in Cracow, Poland on 11-15 September 2010. This biennial meeting aims to bring together researchers and practitioners in the field of natural computing. Natural Computing is the study of computational systems, which use ideas and get inspiration from natural systems, including biological, ecological, physical, chemical, and social systems. It is a fast-growing interdisciplinary field, in which a range of techniques and methods are studied for dealing with large, complex, and dynamic problems with various sources of potential uncertainties.

PPSN XI will be a showcase of a wide range of topics in Natural Computing including, but not restricted to: Evolutionary Computation, Neural Computation, Molecular Computation, Quantum Computation, Artificial Life, Swarm Intelligence, Artificial Ant Systems, Artificial Immune Systems, Self-Organizing Systems, Emergent Behaviors, and Applications to Real-World Problems. PPSN XI will also feature workshops and tutorials covering advanced and fundamental topics in the field of natural computation.

All accepted papers will be presented during poster sessions and will be included in the proceedings. Following the tradition of PPSN, proceedings will be published in the Series Lecture Notes in Computer Science (LNCS) by Springer.

Paper Presentation Following the now well-established tradition of PPSN conferences, all accepted papers will be presented during small poster sessions of about 16 papers. Each session will contain papers from a wide variety of topics, and will begin by a plenary quick overview of all papers in that session by a major researcher in the field. Past experiences have shown that such presentation format led to more interactions between participants and to a deeper understanding of the papers. All accepted papers will be published in the Proceedings.

General Chair

Robert Schaefer (AGH, Cracow, PL)

Honorary Chair

Hans-Paul Schwefel (Tech. Universität Dortmund, DE)

Program Co-Chairs

Carlos Cotta (University of Malaga, ES)
Joanna Kolodziej (University of Bielsko-Biala, PL)
Günter Rudolph (Tech. Universität Dortmund, DE)

Tutorials Chair

Krzysztof Cetnarowicz (AGH, Cracow, PL)

Workshop Chair

Aleksander Byrski (AGH, Cracow, PL)

Important dates

Workshop Proposals Submission	January 3, 2010
Workshop Proposals Notification	February 19, 2010
Paper Submission	April 6, 2010
Author Notification	May 21, 2010
Papers Camera Ready Submission	June 11, 2010
Early Registration	June 11, 2010
Conference	September, 11-15, 2010

Seventh International Conference on Swarm Intelligence

September 8-10, 2010. Brussels, Belgium

Homepage: http://iridia.ulb.ac.be/ants2010

Deadline February 28, 2010

Swarm intelligence is a relatively new discipline that deals with the study of self-organizing processes both in nature and in artificial systems. Researchers in ethology and animal behavior have proposed many models to explain interesting aspects of social insect behavior such as self-organization and shape-formation. Recently, algorithms and methods inspired by these models have been proposed to solve difficult problems in many domains.

An example of a particularly successful research direction in swarm intelligence is ant colony optimization, the main focus of which is on discrete optimization problems. Ant colony optimization has been applied successfully to a large number of difficult discrete optimization problems including the traveling salesman problem, the quadratic assignment problem, scheduling, vehicle routing, etc., as well as to routing in telecommunication networks.

Another interesting approach is that of particle swarm optimization, that focuses on continuous optimization problems. Here too, a number of successful applications can be found in the recent literature. Swarm robotics is another relevant field. Here, the focus is on applying swarm intelligence techniques to the control of large groups of cooperating autonomous robots.

ANTS 2010 will give researchers in swarm intelligence the opportunity to meet, to present their latest research, and to discuss current developments and applications.

The three-day conference will be held in Brussels, Belgium, on September 8-10, 2010. Tutorial sessions will be held in the mornings before the conference program.

Relevant Research Areas

ANTS 2010 solicits contributions dealing with any aspect of swarm intelligence. Typical, but not exclusive, topics of interest are:

- Behavioral models of social insects or other animal societies that can stimulate new algorithmic approaches.
- Empirical and theoretical research in swarm intelligence.
- Application of swarm intelligence methods, such as ant colony optimization or particle swarm optimization, to real-world problems.
- Theoretical and experimental research in swarm robotics systems.

Publication Details As for previous editions of the ANTS conference, proceedings will be published by Springer in the LNCS series (to be confirmed). The journal Swarm Intelligence will publish a special issue dedicated to ANTS 2010 that will contain extended versions of the best research works presented at the conference.

Best Paper Award

A best paper award will be presented at the conference.

Further Information

Up-to-date information will be published on the web site http://iridia.ulb.ac.be/ants2010/. For information about local arrangements, registration forms, etc., please refer to the above-mentioned web site or contact the local organizers at the address below.

Conference Address

ANTS 2010
IRIDIA CP 194/6 — Tel +32-2-6502729
Université Libre de Bruxelles — Fax +32-2-6502715
Av. F. D. Roosevelt 50 — http://iridia.ulb.ac.be/ants2010
1050 Bruxelles, Belgium — email: ants@iridia.ulb.ac.be

Important Dates

Submission deadline	March 28, 2010
Notification of acceptance	April 30, 2010
Camera ready copy	May 14, 2010
Conference	September 8–10, 2010

January 2011

FOGA 2011 - Foundations of Genetic Algorithms

January 5-9, 2011, Schwarzenberg, Austria

Homepage: http://www.sigevo.org/foga-2011

Enquiries and Submissions: foga@fhv.at

Deadline Monday July 5, 2010

We invite submissions of extended abstracts for the eleventh Foundations of Genetic Algorithms workshop. FOGA is only held every two years and focuses on theoretical foundations of all flavors of evolutionary computation. It will next be held in the Gasthof Hirschen hotel in Schwarzenberg in Austria from Wednesday, January 5 to Sunday January 9, 2011. Prof. Dr. Karl Sigmund has agreed to deliver a keynote lecture. Attendance is limited to people who submitted papers, or those requesting attendance in advance. Students are particularly encouraged to participate.

Submissions should address theoretical issues in evolutionary computation. Papers that consider foundational issues, place analysis in the wider context of theoretical computer science, or focus on bridging the gap between theory and practice are especially welcome. This does not prevent the acceptance of papers that use an experimental approach, but such work should be directed toward validation of suitable hypotheses concerning foundational matters.

Extended abstracts should be between 10-12 pages long. To submit, please email a compressed postscript or a PDF file to foga@fhv.at no later than Monday, July 5, 2011. In your email, also include the title of the paper, and the name, address and affiliation of all the authors. Submitted papers should use standard spacing and margins, with 11pt or 12pt font for the main text. Authors using LaTeX should either use the standard article style file or the FOGA style file which can be found at the conference web-site. To ensure the reviews are double-blind authors are asked to remove references to themselves from their paper.

Notification will be September 13, 2011 and drafts of the full paper will be needed by December 6, 2010. These drafts will be distributed as part of a preprint to participants at FOGA. Authors of papers presented at the FOGA workshop will be asked to contribute final versions of their papers (based on discussion/feedback at the meeting) as part of the final volume.

Important Dates

Extended abstracts due	July 5, 2010
Notification to authors	September 13, 2010
Registration and room booking deadline	October 8, 2010
Pre-proceedings camera ready manuscript due	December 6, 2010
FOGA workshop	January 5–9, 2011
Post workshop proceedings	February 21, 2011

Organizers

Prof. Dr. habil. Hans-Georg Beyer	www2.staff.fh-vorarlberg.ac.at/ hgb/
Dr. W. B. Langdon	www.dcs.kcl.ac.uk/staff/W.Langdon/

Further Information

Enquiries and submissions: foga@fhv.at

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.