SIGEVOlution

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

Volume 6
Issue 2

in this issue

Editorial

Welcome to the new issue of SIGEVOlution. More than a year has passed from the last editorial and I apologize for the huge delay. I hope you missed "us" since "we" missed you too! Like a phoenix, the mythological long-lived bird that is cyclically regenerated or reborn, here we are again with a new juicy menu. We start with an article by 2013 Humies gold medalist Cameron Browne about LUDI, his framework for evolutionary game design that created Yavalath, the world's first fully computer-generated board game to be commercially released. Then, we have an article about the new Python framework for distributed evolution (DEAP) developed by Francois-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, and Christian Gagné. Our final dessert is a report of the 2013 GECCO competitions that received the astonishing number of 32 submissions.

Unfortunately, I missed the GECCO deadline this year. If you missed it too don't despair since there are still 16 (!) workshops waiting for your contribution (the deadline is March 28, 2014) and several competitions you can join.

As always, I owe my thanks to the many people who helped me in this: Cameron Browne, Francois-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, Christian Gagné, Daniele Loiacono, Cristiana Bolchini, Viola Schiaffonati, Francesco Amigoni, Franz Rothlauf.

The cover photo is a shot of a deluxe Yavalath board by Néstor Romeral Andrés.

See you next year! (just kidding)

Pier Luca
February 18, 2014

SIGEVOlution Volume 6, Issue 2

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

Contributors to this Issue

Cameron Browne
François-Michel De Rainville
Félix-Antoine Fortin
Marc-André Gardner
Marc Parizeau
Christian Gagné

Contents

ISSN: 1931-8499

PPSN 2014

13th International Conference on
Parallel Problem Solving from Nature
September 13-17, 2014
Ljubljana, Slovenia
http://ppsn2014.ijs.si

General Chair
Bogdan Filipič, Jožef Stefan Institute, Slovenia

Honorary Chair
Hans-Paul Schwefel, Dortmund UT, Germany

Program Chairs
Thomas Bartz-Beielstein
Cologne University of Applied Sciences, Germany
Jürgen Branke, University of Warwick, UK
Jim Smith, University of the West of England, UK

Tutorial Chairs
Shih-Hsi "Alex" Liu, California State University, Fresno, USA
Marjan Mernik, University of Maribor, Slovenia

Workshop Chairs
Evert Haasdijk, VU University Amsterdam, The Netherlands
Tea Tušar, Jožef Stefan Institute, Slovenia

Publications Chair
Jurij Šilc, Jožef Stefan Institute, Slovenia

Local Organizing Committee Chair
Gregor Papa, Jožef Stefan Institute, Slovenia

AIMS AND SCOPE

The 13th International Conference on Parallel Problem Solving from Nature (PPSN XIII) will be organized by the Jožef Stefan Institute, Ljubljana, Slovenia, and held at the Ljubljana Exhibition and Convention Centre on September 13-17, 2014. The conference aims to bring together researchers and practitioners in the field of Natural Computing. Natural Computing is the study of computational systems that 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.

PAPER PRESENTATION

Following the well-established tradition of PPSN conferences, all accepted papers will be presented during poster sessions. Each session will contain several papers, 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 deeper understanding of the papers. All accepted papers will be published in the proceedings as a volume of the Lecture Notes in Computer Science (LNCS) Springer series. The format should follow the LNCS style (http://www.springer.de/comp/lncs/authors.html). Prospective authors are invited to contribute their high-quality original results in the field of Natural Computing as papers of no more than 10 pages.

CONTACTS

Email: ppsn2014@ijs.si
Twitter: twitter.com/ppsn2014

IMPORTANT DATES

March 17, 2014 Paper submission
May 19, 2014 Author notification
June 2, 2014 Camera-ready paper submission
June 4, 2014 Early registration
September 13-17, 2014 Conferencee

Evolutionary Game Design

Automated Game Design Comes of Age

Cameron Browne

The "Humies" awards are an annual competition held in conjunction with the Genetic and Evolutionary Computation Conference (GECCO), in which cash prizes totalling $10,000 are awarded to the most human-competitive results produced by any form of evolutionary computation published in the previous year. This article describes the gold medal-winning entry from the 2012 "Humies" competition, based on the LUDI system for playing, evaluating and creating new board games. LUDI was able to demonstrate human-competitive results in evolving novel board games that have gone on to be commercially published, one of which, Yavalath, has been ranked in the top 2.5% of abstract board games ever invented. Further evidence of human-competitiveness was demonstrated in the evolved games implicitly capturing several principles of good game design, outperforming human designers in at least one case, and going on to inspire a new subgenre of games.

Introduction

General game playing (GGP) involves the development of systems for playing a range of games well rather than any single game expertly. GGP was first proposed in the late 1960s [12] but has only recently emerged as a field in its own right, mainly due to the annual AAAI GGP competitions [7], as AI researchers see the benefits of such general approaches to machine intelligence. General game players are improving each year in strength and generality, but an important aspect of the games being modelled has largely gone ignored, namely how good are they? In 1992, Barney Pell observed [11]:

> "If we could develop a program which, upon consideration of a particular game, declared the game to be uninteresting, this would seem to be a true sign of intelligence! So when this becomes an issue, we will know that the field has certainly matured."

In 2007 I developed a program called LUDI that was not just a general game player, but a complete general game system (GGS) for playing, evaluating and generating new games. LUDI answered Pell's challenge by demonstrating how combinatorial games could be automatically measured for their potential to interest human players, but also went a step further to show how this information could be used to direct the evolutionary search for new high quality games. Two of the games produced by LUDI have since been commercially published, and one of them, Yavalath, has proved to be particularly successful with players. These were the first – and to my knowledge remain the only – examples of fully computer-generated games to be commercially published.

LUDI, and the games it produced, won the gold medal at the 2012 Genetic and Evolutionary Computation Conference (GECCO) "Humies" awards, for human-competitive results in evolutionary computation [9]. This article is an expanded version of the winning presentation made at the 9th annual "Humies" competition, held at GECCO in Philadelphia in July 2012. It briefly describes the inner workings of LUDI, how it evaluates and creates new games, describes its two best games in detail, then concludes by summarising the evidence for human-competitiveness.

Representation

Games are modelled as rule trees and described as LISP-like symbolic expressions or s-expressions. For examples, Figures 1 and 2 show the rule tree and corresponding s-expression for the game of Tic-Tac-Toe. This representation has the advantage of being high-level, structured and human-readable; the average player should be able to read through this set of rules and at least recognise this game (once i-nbors is understood to mean "diagonal neighbors"). This LUDI game description language (GDL) was primarily intended to model how human designers might conceptualise games, as structured relationships of rules.

An advantage of using a grammar-based approach is that the resulting rule trees possess an inherent modularity. For example, all rules pertaining to the board are contained in the subtree below the board node, all movement rules pertaining to a particular piece are contained in the subtree below that piece's defining node, and so on. This makes it easier to swap movement rules between pieces, pieces between games, board types between games, and so on, which are operations typically employed by human designers in devising new games.

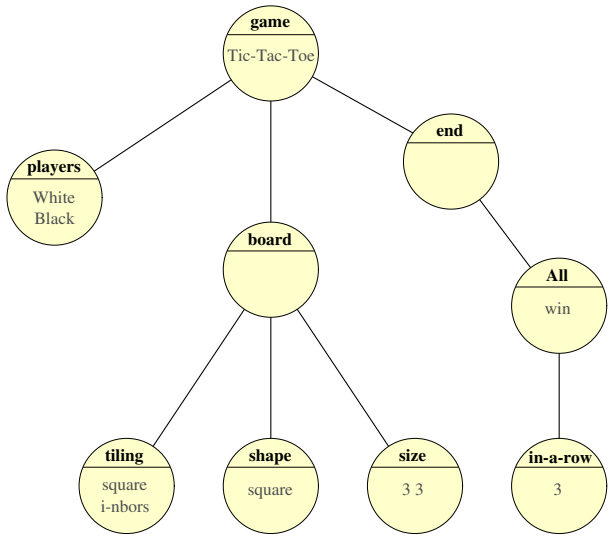


Fig. 1: Rule tree for Tic-Tac-Toe.

```
(game Tic-Tac-Toe
   (players White Black)
   (board
      (tiling square i-nbors)
      (shape square)
      (size 3 3)
   )
   (end (All win (in-a-row 3)))
)
```

Fig. 2: S-expression for Tic-Tac-Toe.

Operation

New games are created by a straightforward evolutionary approach, using standard genetic programming (GP) operators of crossover and mutation [8] to mate pairs of games selected by roulette selection from a population of individuals. Figure 3 shows a diagrammatic summary of this process.

Game design is very much a process of innovation followed by refinement, in which the designer has the great idea for a new mechanism, then must find the optimal combination of rules to realise this mechanism through a series of micro-iterations. The GP operators seem especially well-suited to this task, as crossover provides innovation and mutation provides refinement, although in practice there will be some overlap between the two. The game descriptions are strongly typed [10], so that rules are only crossed over with compatible rules, which maintains modularity and facilitates the creation of meaningful rule combinations.

New game names are also automatically generated by the system, using a Markov chain approach based on n-gram (letter combination) frequencies found in a list of Tolkien-style character names.

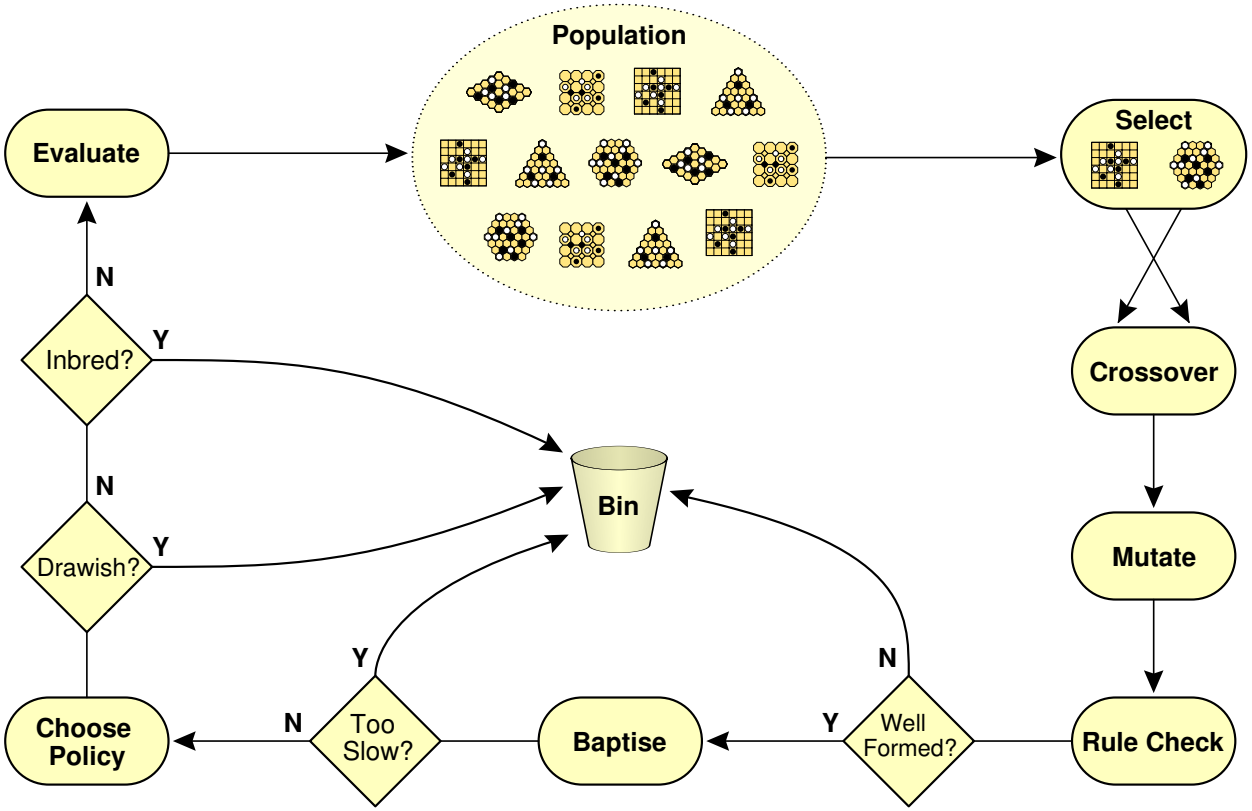


Fig. 3: Game design process.

Game Fitness

Evaluating games for their potential to interest human players is a much more difficult task. This is where the bulk of the work was done on LUDI, and represents its most important contribution. Boumanza [3] asks the questions posed by most people who encounter the project:

> "How can computers create interesting games? What makes a game interesting and how can a machine decide if a game is of interest to human players?"

Game fitness is measured through self-play trials, as the true nature of a game cannot be observed in its rules alone, but emerges during play. LUDI plays each game between two AI opponents a number of times, and measures certain trends observed during play. For example, Figure 4 is a lead plot over the course of a game, in which the bold (red) line shows the estimated difference in board position between the eventual winner (White) and loser (Black) with each move. The board position estimates are provided by the AI players as a by-product of the search.

The aesthetic criterion being measured in Figure 4 is drama, which is the degree to which the player currently in a negative (losing) position can turn the tables to win the game. Such drama keeps a game interesting for both players. The example in Figure 4 shows quite a dramatic game, in which White spends a number of moves in a relatively negative position, before eventually retaking the lead to win. 57 such aesthetic criteria were defined for LUDI, of which 17 were found to give a strong correlation with human player rankings for a database of 79 source games [5].

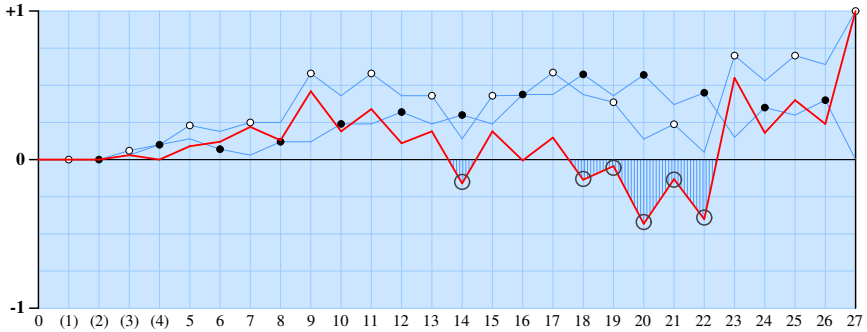


Fig. 4: Lead plot showing drama.

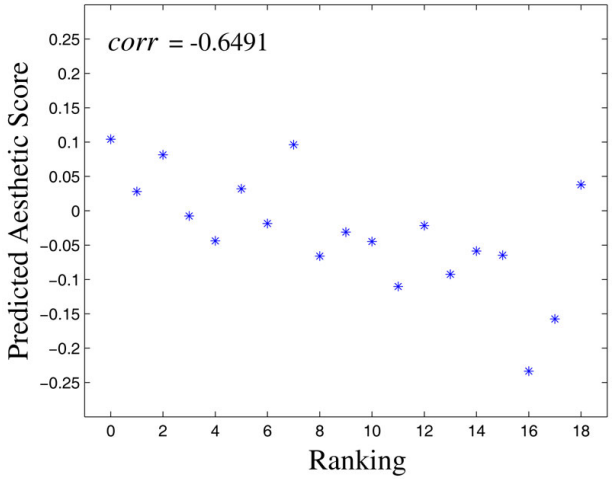


Fig. 5: Estimated fitness versus player ranking.

Results

Starting with this database of 79 source games, the evolutionary process was run over approximately three weeks to yield 1,379 evolved games of varying quality. 19 of these games were deemed to be playable, according to a simple playability filter based on:

1. Completion: Games produced a result more often than not.
2. Bias/balance: Games did not unduly favour either player or colour.
3. Game length: Games were not too short nor too long.

Game length, in particular, proved to be a surprisingly effective indicator of flawed games, as also found by Althöfer [2], filtering out over 95% of evolved individuals with minimal false positives. It is a powerful metric as it detects trivial games that end in a few moves, indecisive games that go on for too long, and impossible games whose winning conditions can never be achieved.

Figure 5 shows the reasonably strong correlation between the estimated aesthetic score for each game, and its actual ranking by human play testers. The aesthetic criteria proved useful in predicting an evolved game's potential to interest human players.

Yavalath

Yavalath is the most successful game evolved by LUDI. It was ranked #4 by the system and #2 by human play testers. The complete rule set is given in Figure 6.

Players take turns placing a piece of the their colour on the board (Figure 7), and win by making a line of four of their colour, but lose by making a line of three of their colour beforehand. The rules are not much more complex than those for Tic-Tac-Toe, but this apparent simplicity hides a surprising emergent twist that subverts the familiar N-in-a-row genre and makes Yavalath quite an interesting game.

Analysis

Consider the situation shown in Figure 8, with White to play. If White plays move 1 as shown (Figure 9), then Black is forced to reply with blocking move 2, which unfortunately for them creates a line of three to lose the game; White's move 1 forced a win. This mechanism of forcing moves constitutes the heart of game, and allows clever sequences of play that can manipulate the opponent into a disadvantageous and eventually losing position.

The situation shown in Figure 10 is a more complex puzzle, with White to play. Note that if Black is allowed to play at either move X then White is forced to reply in the other cell X and complete a losing line of three. White therefore must play a forcing move next turn to stop Black doing this. There are three forcing moves available to White, marked *a*, *b* and *c*. Figure 11 shows White's only winning sequence from move *c*; any other move (or set of moves) would lose this game.

```
(game Yavalath
    (players White Black)
    (board
        (tiling hex)
        (shape hex)
        (size 5)
    )
    (end
        (All win (in-a-row 4))
        (All lose (in-a-row 3))
    )
)
```

Fig. 6: Evolved rules for Yavalath.

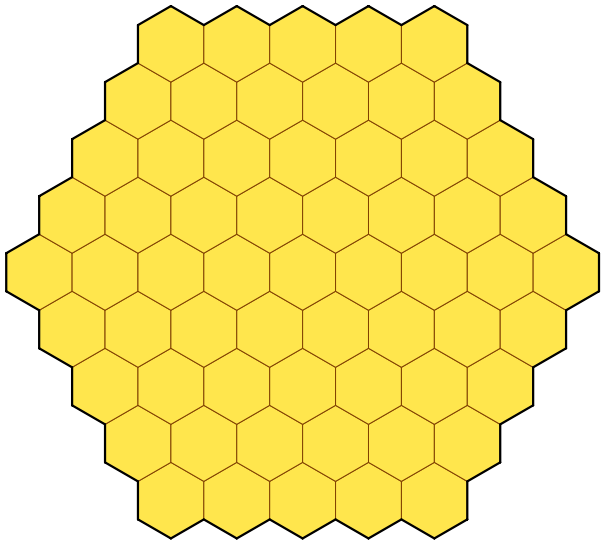


Fig. 7: Yavalath board (starts empty).

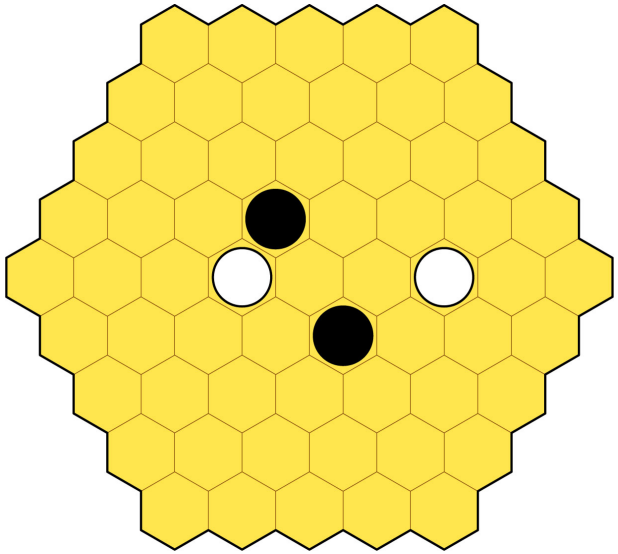


Fig. 8: White to play.

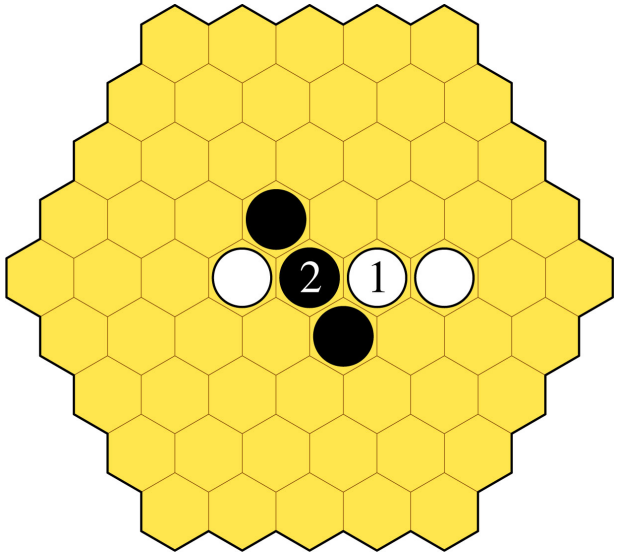


Fig. 9: White move 1 forces losing reply 2.

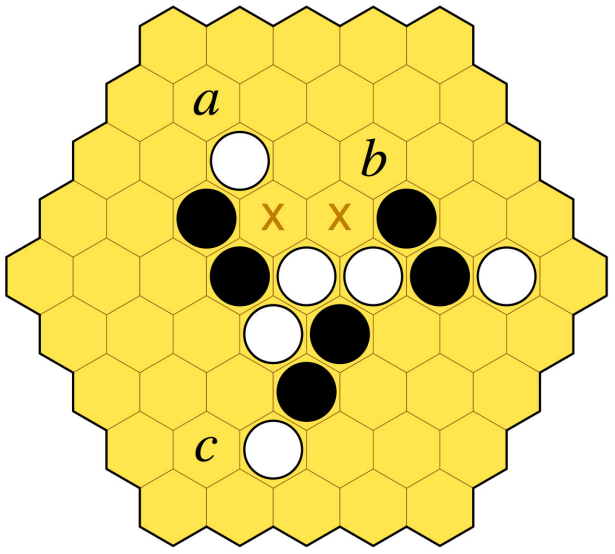


Fig. 10: White to play.

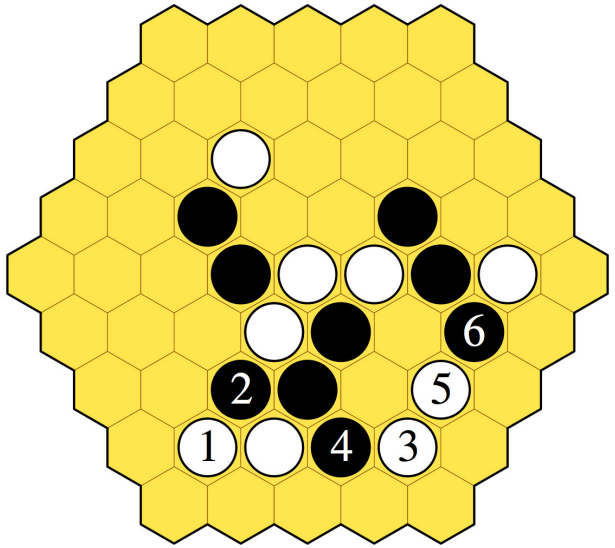


Fig. 11: White's only winning sequence.

Yavalath therefore allows the creation of interesting puzzles in general play, one of Thompson's [15] requirements for an abstract board game to have lasting interest for players.

Such complex behaviour emerging from simple rules is what defines the best abstract board games – a minute to learn, a lifetime to master – and is exactly the sort of phenomenon we were hoping to see emerge from the evolutionary search.

Quality

A requirement of the "Humies" competition is that submissions must satisfy an "arms length" standard. That is, the quality of the results must be demonstrated through an independent source, other than the opinion of the author or their colleagues, such as publication in a peer-reviewed journal or entry in an expert database. This posed a bit of a problem for LUDI and its output, as games are an artistic as much as a scientific result (even though they can still be measured empirically), and there is no internationally recognised panel of experts for game design. However, the online board game database BoardGameGeek (BGG) provides a useful source of peer review, and the equivalent of an expert database in this field.

Board Game Geek (BGG) [1] is the world's foremost online community of board game players, designers and publishers. It has over 400,000 members including many of the world's best game designers and players, and its database lists every known board game (over 45,000). Games submitted to BGG must pass a review and moderation process before being added to the database, and its peer voting mechanism for rating database entries is hidden from casual users, which encourages educated assessments, and is carefully policed. It is generally acknowledged that the BGG ranking for a game is a fair assessment of its value in the eyes of players.

Yavalath was added to the BGG database in the "Abstract Games" category, which contains more than 4,300 games. It achieved a high ranking of #99 in this category in August 2011, placing it in the top 2.5% of abstract board games ever invented, according to the BGG community.

Fig. 12: The Yavalath Deluxe set from nestorgames.

To put this in context, Yavalath was ranked below Chess and Go, but above many well known and respected games, several of which have won awards and been recognised as games of high quality:

#3	Go
#45	Chess
#99	**Yavalath**
#112	Backgammon
#140	Abalone
#267	Othello
#539	Mastermind
#546	Chinese Checkers

Shortly before giving the "Humies" presentation, I had the humbling realisation that Yavalath is also ranked above all of my own more than 60 games on BGG. Players appear to prefer my programme's games to my own.

Publication

Yavalath was published in 2009 by independent games publisher nestorgames [13]. It was the first game in their catalogue, which has grown to almost 100 games in three years, and remains a flagship product. Figure 12 shows the Yavalath Deluxe set, released as a special edition for nestorgames' third anniversary, worth around $200.

Note that this picture shows the three-player version of the game; Yavalath is one of the few board games that plays equally well with two or three players, another attractive marketing point.

Yavalath is still the first game that nestorgames owner Néstor Romeral Andrés teaches to new players at exhibitions and conventions, as the simple rules mean that players can start playing it immediately, but the emergent twist surprises them and makes the game addictive as they replay it to explore this twist more fully. There is an "aha!" moment when players discover the forced move mechanism – Romeral Andrés reports that most players do actually say "aha!" at this point (or the Spanish equivalent "anda!") – and it was satisfying to hear more than one audible "aha!" coming from the "Humies" audience when describing the game.

Romeral Andrés points out that players are generally surprised when it is revealed that Yavalath was designed by computer. The popular consensus among players appears to be that they would have expected a computer-generated game to be more complex and difficult to play, and less fun.

Inspiration

Immediately following the public release of Yavalath, an experienced game designer announced that he had devised a similar "5 but not 4" mechanism on the (square) Go board, but never got it working satisfactorily and had shelved the idea years ago. While he quipped that the idea had come from his "human brain", it was LUDI's automated search that found a rule set to successfully realise the "N but not N-1" mechanism. In terms of the game design process, both had made the innovation, but only LUDI had successfully refined it.

Yavalath has since inspired the invention of a number of further games, including: Tritt, Cross, Tailath, Morro, Coffee, Epsilon and Manalath. The basic "N but not N-1" mechanism has been applied to other game types, including:

1. Form a group of size N but not size N-1.
2. Connect N sides of the board but not N-1 sides.

The basic Yavalath mechanism has therefore been extrapolated to a more general "N but not sub(N)" rule in a number of contexts, constituting a new sub-genre of games (tentatively called *subset games*).

Game Design Principles

In creating Yavalath, LUDI implicitly captured some fundamental principles of good game design used by human designers, albeit unintentionally. These principles were not programmed into LUDI, but emerged as a serendipitous by-product of the evolutionary search.

Familiarity with Novelty. Players generally prefer games with familiar rules and short learning curves, but will quickly become bored with games that do not offer something new. Achieving these competing objectives of familiarity and novelty in a single game is a significant challenge for designers, but LUDI did exactly that with Yavalath. Further, the fact that the winning and losing conditions both use the same (N-in-a-row) basis means that players need to process less information to understand and remember the rules, maximising the game's clarity as they mentally plan their moves.

Rule Tension. The inclusion of rules that create strategic conflict against other rules is a common mechanism for increasing tension in a game, as players must carefully weigh the benefits of a given move with any detrimental side-effects. Yavalath's "4 but not 3" mechanism is an obvious manifestation of this principle, as lines of a certain length are infinitely beneficial while lines of another length are fatal; players cannot simply extend their best line each turn, but must find the optimal move in the face of these multiple and competing objectives.

Ndengrod (aka Pentalath)

Ndengrod, the second most successful game evolved by LUDI, was also published by nestorgames in 2009 [14]. Its name was changed to Pentalath for release; just like the automatically generated games, some automatically generated names did not work as well as others. Ndengrod was ranked the #1 evolved game by both the system and human play testers, but remains less popular than Yavalath due to its more complex rule set (Figure 13). Players take turns placing a piece of the their colour on the board, capturing any surrounded enemy groups (as per Go), and win by making a line of five pieces of their colour. Ndengrod has a much steeper learning curve than Yavalath. Firstly, players must come to grips with the surround capture rule, and secondly, it can take several games before a player begins to grasp the underlying strategy, as Ndengrod is surprisingly subtle and deep.

Analysis

Figure 15 shows a typical passage of play, by way of example. The position on the left shows Black to play and attempt to save their threatened group. If Black chases the only available freedom with moves 1, 3 and 5, then White can make the replies shown (2 and 4) to force a ladder down the side (middle). A Go player might do this with the intention of running the threatened black group into the bottom edge to capture it. However, Black has now made a line of four pieces, so White is forced to immediately block this line with move 6 (right). Black is able to escape the ladder with move 7, and in fact now has the upper hand, as capturing the stray white piece 6 and completing their line of five should not be too difficult. White made a serious mistake in following this ladder.

Ndengrod is actually very much a connection game [4] beneath the surface. It is most important to establish safe (two-eyed) groups and connect them together across the board, while the 5-in-a-row goal is more of a local tactical concern during this process. It is possible to win the game at any stage by forcing 5-in-a-row against a careless opponent, but in games between experienced and careful players the board will tend to fill up before this happens. In such cases, the game takes on a territorial aspect and will typically be won by the player with the strongest groups that contain the most freedoms (eyes).

```
(game Ndengrod
  (players White Black)
  (board
    (tiling hex)
    (shape trapezium)
    (size 7)
  )
  (pieces (Piece All (moves (move
    (pre (empty to))
    (action push)
    (post (capture surround))
  ))))
  (end (All win (in-a-row 5)))
)
```

Fig. 13: Evolved rules for Ndengrod.

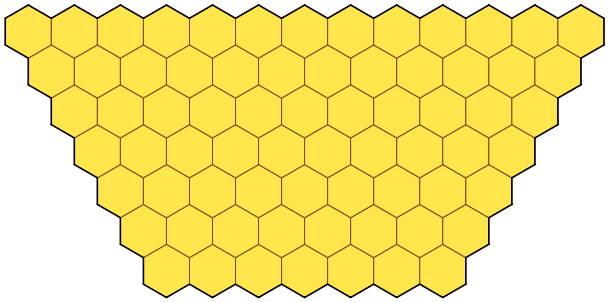


Fig. 14: Ndengrod board (starts empty).

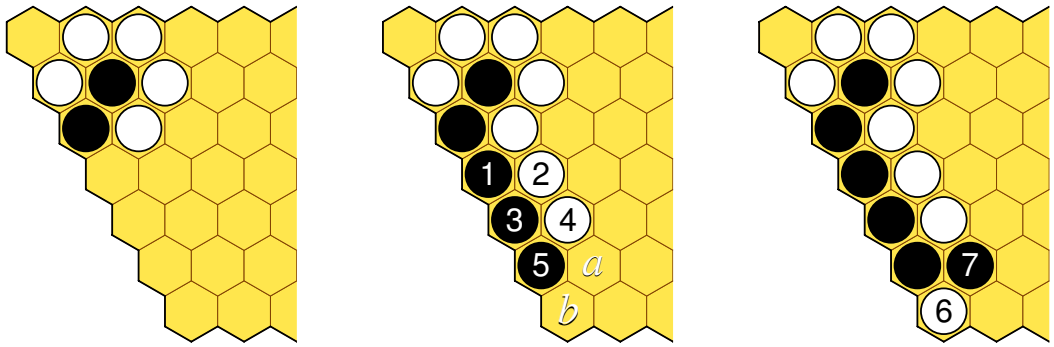


Fig. 15: White forces a ladder but is forced to abandon it, giving Black the upper hand.

The fact that players may not pass, unlike Go, introduces a beautiful self-correcting mechanism to the end game. Players will eventually be forced to fill in their own eyes if not able to play elsewhere, making it inevitable that at least one capture will occur before the board fills up completely, following which the opponent will usually win the game quickly. Players must prepare for this end game well before it occurs, and ensure that they can outlast their opponent in such an eye-filling race.

Game Design Principles

In creating Ndengrod, LUDI also implicitly captured some principles of good game design used by human designers, but in this case more specific to the geometry of the game.

Use the Geometry (Board Shape). Ndengrod was originally specified by LUDI on the trapezoidal board shown in Figure 14, but was remapped to the hexagonal Yavalath board for publication, so that both games could be released as a set. The name Ndengrod was also changed to Pentalath for publication.

However, after years of testing and player feedback, it became clear that Ndengrod is a better game on its original trapezoidal board, for a number of reasons:

1. The presence of both acute and obtuse board corners allows greater variety in tactical play.
2. The longer board rows allow greater scope for 5-in-a-row threats.
3. The board is less symmetric, being wider than high, encouraging greater variety in tactical play.
4. The board is larger (70 vs 61 cells) allowing more complex global connection battles.

Ndengrod has now been re-released on the original trapezoidal board [14]. LUDI had found what appears to be the optimal geometry for this game, which was only worsened by our subsequent modifications; LUDI knew better than us in this case.

Use the Geometry (No Ko on Hexagonal Grids). The ko rule, used in Go, specifies that players cannot make a move that would repeat the previous board position. For example, Figure 16 shows a Go position in which White captures a black piece, but Black is not allowed to immediately recapture at point X as this would repeat the previous board state. The ko rule is essential in Go for avoiding infinite cycles of capture and recapture.

One of the mysteries of Ndengrod is why the game works so well without a ko rule, even though it features Go-like surround capture. This question remained unanswered until recently, when it was (re)discovered that ko situations simply don't occur on the hexagonal grid, due to its geometry [6]. For example, Figure 17 shows the analogous position on the hexagonal grid, in which White captures a single surrounded black piece, but in this case Black cannot possibly recapture it on the next move. This is because on the square grid groups are simply (i.e. orthogonally) connected whereas the surrounding enemy set need not be, but on the hexagonal grid both the group and its surrounding enemy set are simply connected. LUDI had implicitly captured this knowledge that the ko rule is not needed on the hexagonal grid, in its choice of rules for Ndengrod. This resulted in a simpler, more efficient rule set, and taught me a fundamental fact regarding Go-like game design that I did not previously know. In creating Ndengrod, LUDI had devised one of the very few Go-like games that actually works well on the hexagonal grid.

Human Competitiveness

Entries in the "Humies" competition are judged for human competitiveness according to eight key criteria [9]. I claimed that LUDI and the games it produced satisfied criteria C), D) and F), as follows:

C) The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts. Yavalath was placed in the BGG database and ranked in the top 2.5% of abstract board games ever invented, placing it above many famous and popular games.

D) The result is publishable in its own right as a new scientific result — independent of the fact that the result was mechanically created. Yavalath and Ndengrod have been commercially published for over three years, and Yavalath remains a flagship product for nestorgames.

F) The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered. Yavalath has been ranked on BGG above many popular games that have won awards and were considered achievements at their time of release, including Othello, Mastermind, Abalone, and others.

Cases could have been made for other criteria. For example, Yavalath and Ndengrod could be patented (criterion A) although this would not be cost effective, and LUDI did not "solve a problem of indisputable difficulty in its field" (criterion G) so much as find a range of good solutions for that problem (there is no single "best" game). Such claims would have been tenuous and were not made.

Conclusion

In summary, LUDI was able to:

- Evolve new and interesting games.
- Inspire a new sub-genre of games.
- Find good solutions to problems encountered by human designers.
- Implicitly capture several principles of good game design.
- Yield insights into game design that have surprised me.
- Produce a game that has proven more popular among players than my own games.

These results apparently satisfied the criteria for human competitiveness, and impressed the "Humies" judges sufficiently to win the gold medal. I believe that a large part of this success was due to the novel and creative nature of the problem; game design is an art as much as a science, and a very human craft. Modelling the underlying principles computationally is a very difficult task, and one that had not been previously achieved to the level demonstrated by LUDI and its creations. In future, we can expect to see a greater number of increasingly human-competitive results in game design and other creative tasks, as researchers increasingly turn their attention to procedural content generation [16] in such areas.

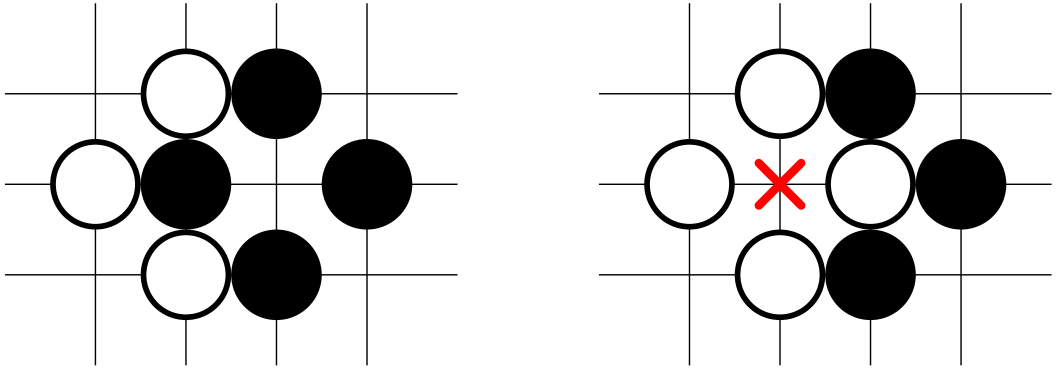


Fig. 16: The ko rule forbids Black from immediately recapturing in Go.

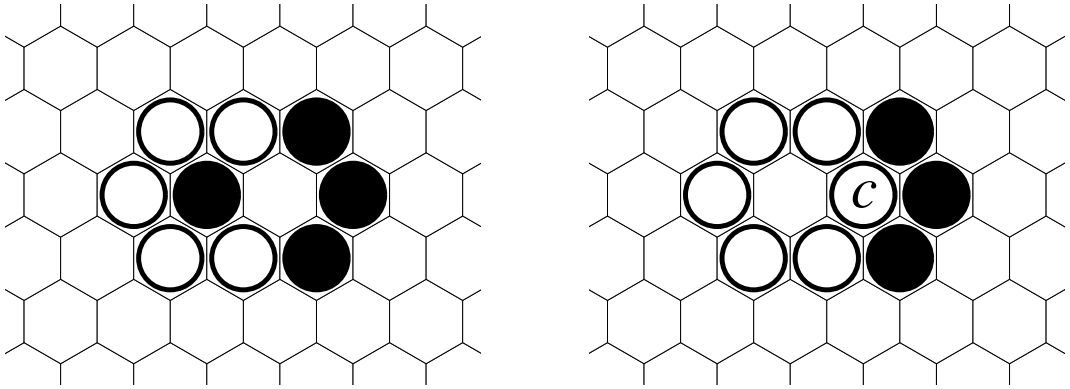


Fig. 17: The ko rule is not needed on the hexagonal grid.

References

[1] S. Alden (2000) "*BoardGameGeek*", http://www.boardgamegeek.com

[2] I. Althöfer (2003) *Computer-aided game inventing*, Technical Report, Friedrich Schiller Universität Jena.

[3] A. Boumanza (2012) "*Cameron Browne: Evolutionary Game Design*", Genetic Programming and Evolvable Machines, 13:3, 407-9.

[4] C. Browne (2005) *Connection Games: Variations on a Theme*, AK Peters, Massachusetts.

[5] C. Browne (2011) *Evolutionary Game Design*, Springer, Berlin.

[6] C. Browne (2012) "*Go Without Ko on Hexagonal Grids*", ICGA Journal, 35:1, 37-40.

[7] M. Genesereth, N. Love and B. Pell (2005) *"General Game Playing: Overview of the AAAI Competition"*, AI Magazine, 26:2, 62-72.

[8] J. Koza (1992) *Genetic Programming*, MIT Press, Cambridge.

[9] J. Koza (2012) *"The Annual 'Humies' Awards – 2004-2012"*, http://www.genetic-programming.org/combined.html

[10] D. Montana (1995) *"Strongly typed genetic programming"*, Journal of Evolutionary Computation, 3:2, 199-230.

[11] B. Pell (1992) "*METAGAME in symmetric Chess-like games*", Heuristic Programming in Artificial Intelligence 3, eds. H. Van den Kerik and L. Allis, Ellis Horwood, Chichester.

[12] J. Pitrat (1968) *"Realisation of a general game-playing program"*, IFIP Congress, 2, 1570–1574.

[13] N. Romeral Andrés (2012a) "*Yavalath*", http://www.nestorgames.com/#yavalathdeluxe_detail

[14] N. Romeral Andrés (2012b) "*Pentalath*", http://www.nestorgames.com/#pentalath_detail

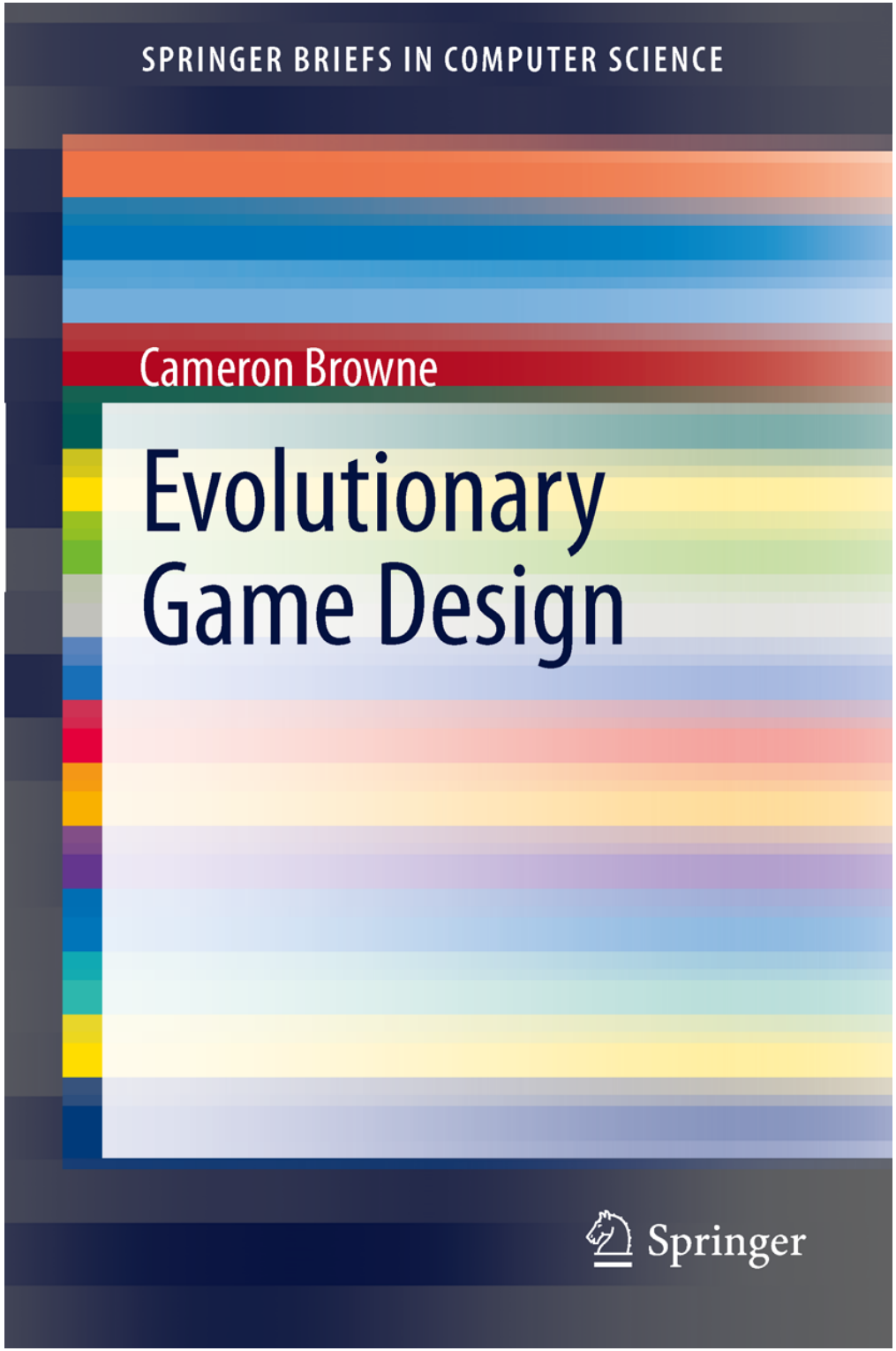
[15] M. Thompson (2000) *"Defining the abstract"*, The Games Journal, http://www.thegamesjournal.com

[16] J. Togelius, G. Yannakakis, K. Stanley and C. Browne (2011) *"Search-based Procedural Content Generation: A Taxonomy and Survey"*, IEEE Trans. on Computational Intelligence and AI in Games, 3:3, 172-186.

About the authors

Cameron Browne received the Ph.D. degree in Computer Science from the Queensland University of Technology (QUT), Brisbane, Australia, in 2008, winning the Dean's Award for Outstanding Thesis. He was Canon Research Australia's Inventor of the Year for 1998, and won the 2012 GECCO "Humies" gold medal for human-competitive results in evolutionary computation. He is the author of the books Hex Strategy, Connection Games and Evolutionary Game Design, and is an Associate Editor of the IEEE Transactions on CI and AI in Games (TCIAIG). Dr. Browne is currently a Research Fellow at Goldsmiths College, University of London, and will soon be taking up a position as Senior Research Fellow at the Queensland University of Technology (QUT) in Brisbane, Australia.

Homepage: http://www.cameronius.com
Email: cameron.browne@btinternet.com



Evolutionary Game Design
Cameron Browne
Springer, 2011
ISBN 978-1-4471-2178-7

This book tells the story behind Yavalath, the world's first fully computer-generated board game to be commercially released. It is based on the PhD thesis "Automatic Generation and Evaluation of Recombination Games", which describes the development of a software system called LUDI able to play, evaluate and evolve new games from existing rule sets. The LUDI project demonstrated how board games can be empirically measured for their potential to interest human players, and how this knowledge can be used to direct the automated search for new and interesting games.

Evolutionary Game Design goes further to place this project in the broader context of computational creativity, and examine questions raised by the creation of Yavalath and its subsequent impact on game players and designers. This book was the main piece of evidence behind the winning entry in the 2012 GECCO "Humies" competition, for human-competitive results in evolutionary computation.

"Evolutionary Game Design is a valuable contribution to evolutionary computation and more generally to artificial intelligence. It is engaging to read, easy to follow and lives up to its promises. Furthermore, it delivers insights that should be helpful to anyone interested in AI and games."

A. Boumanza (2012) on "Evolutionary Game Design" in Genetic Programming and Evolvable Machines, 13:3, 407-9.

DEAP - Enabling Nimbler Evolutions

François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, and Christian Gagné
Département de génie électrique et de génie informatique - Université Laval - Québec (Québec), Canada

DEAP is a Distributed Evolutionary Algorithm (EA) framework written in Python and designed to help researchers developing custom evolutionary algorithms. Its design philosophy promotes explicit algorithms and transparent data structures, in contrast with most other evolutionary computation softwares that tend to encapsulate standardized algorithms using the black-box approach. This philosophy sets it apart as a rapid prototyping framework for testing of new ideas in EA research. An executable notebook version of this paper is available at https://github.com/DEAP/notebooks.

Highlights

- Building blocks for testing ideas
- Rapid prototyping
- Fully transparent
- Parallel ready
- Exhaustively documented
- Available at http://deap.gel.ulaval.ca

Introduction

The DEAP framework [1; 2] is designed over the three following founding principles:

1. Data structures are key to evolutionary computation. They must facilitate the implementation of algorithms and be easy to customize.
2. Operator selection and algorithm parameters have strong influences on evolutions, while often being problem dependent. Users should be able to parametrize every aspect of the algorithms with minimal complexity.
3. EAs are usually embarrassingly parallel. Therefore, mechanisms that implement distribution paradigms should be trivial to use.

With the help of its sister project SCOOP [3] and the power of the Python programming language, DEAP implements these three principles in a simple and elegant design.

Data Structures

A very important part of the success for designing any algorithm — if not the most important — is choosing the appropriate data structures. Freedom in type creation is fundamental in the process of designing evolutionary algorithms that solve real world problems. DEAP's *creator* module allows users to:

- create classes with a single line of code (inheritance);
- add attributes (composition);
- group classes in a single module (sandboxing).

In the following listing, we create a minimizing fitness.

```
from deap import base, creator
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
```

The `create` function expects at least two arguments; the name of the class to be created and the base class it inherits from. The next arguments are used as class attributes. Thus, the class just created is a `FitnessMin` inheriting from the base class `Fitness` and having a `weights` attribute set to the one element tuple $(-1.0,)$, indicating minimization of a single objective. The trailing comma is mandatory to define a single item tuple in Python. A multi-objective fitness would be created using a multi-element tuple.

Next, we define with the same mechanism an `Individual` class inheriting from a `list` and composed with a `fitness` attribute.

```
creator.create("Individual", list, fitness=creator.FitnessMin)
```

When an `Individual` is instantiated, its `fitness` is initialized as an instance of the previously defined `FitnessMin` class. This is illustrated in the following example,

```
ind = creator.Individual([1,0,1,0,1])
ind.fitness.values = (sum(ind),)
```

where an individual is created from a list of binary values and the value of its fitness is set to the sum of its elements. In DEAP, the fitness value is always multi-objective with the single objective case being a tuple of one element.

Operators

Operator selection is another crucial part of evolutionary algorithms. It must be straightforward and its parametrization intuitive. DEAP's *Toolbox* enables users to:

- create aliases for operators;
- register operators' parameters;
- interchange operators efficiently;
- regroup all operators in a single structure.

The next example presents the construction of a toolbox and how operators and their parameters are registered.

```
from deap import tools
toolbox = base.Toolbox()
toolbox.register("mate", tools.cxOnePoint)
toolbox.register("mutate", tools.mutGaussian, mu=0.0, std=1.0)
```

The `register` function expects at least two arguments; the alias of the function and the function itself. The next arguments are passed to the function when called, similarly to the `partial` function from the standard *functools* module. Thus, the first operator is a one point crossover registered under the alias `mate`. The second operator, a gaussian mutation, is registered with its parameters under the generic name `mutate`. Both operators are available from the *tools* module along with many more instruments supporting evolution that are presented at the end of this paper.

During subsequent experiments, replacing the one point crossover by a two point crossover is as easy as substituting the third line of the previous listing by the following one.

```
toolbox.register("mate", tools.cxTwoPoint)
```

Wherever the generic function `mate` is used, the new two point crossover will be used.

Parallelization

DEAP is parallel ready. The idea is to use a mapping operation that applies a function to every item of a sequence, for instance to evaluate the fitnesses. By default, every toolbox is registered with the standard `map` function of Python. For algorithms to evaluate individuals in parallel, one only needs to replace this alias by a parallel map such as the one provided by SCOOP [3], a library capable of distributing concurrent tasks on various environments, from grids of workstations to supercomputers.

```
from scoop import futures
toolbox.register("map", futures.map)
```

DEAP is also compatible with the standard *multiprocessing* module, if the user only cares to run on a single computing node with multiple cores.

```
import multiprocessing
pool = multiprocessing.Pool()
toolbox.register("map", pool.map)
```

With these powerful tools, DEAP allows scientists and researchers with little programming knowledge to easily implement distributed and parallel EAs.

Preaching by Example

The best introduction to evolutionary computation with DEAP is to present simple, yet compelling examples. The following sections set forth how algorithms are easy to implement while keeping a strong grip on how they behave. The first section introduces a classical genetic algorithm and exposes different levels of explicitness. The second section presents how genetic programming is implemented in DEAP and the versatility of the GP module. The final example demonstrates how easy it is to implement a generic distributed island model with SCOOP.

A Simple Genetic Algorithm

A commonly used example in evolutionary computation is the OneMax problem which consists in maximizing the number of ones in a binary sequence. The more ones an individual contains, the higher its fitness value is. Using a genetic algorithm to find such an individual is relatively straightforward. Applying crossovers and mutations on a population of randomly generated binary individuals and selecting the fittest ones at each generation usually converge to a perfect (all ones) solution. A problem of this simplicity should be solved with a very simple program.

Figure 1(a) presents all that is needed to solve the OneMax problem with DEAP. The first two lines import the necessary modules. Next, on lines 3 and 4, two types are created; a maximizing fitness (note the positive weights), and a list individual composed with an instance of this maximizing fitness. Then, on lines 5 and 6, the evaluation function is defined. It counts the number of ones in a binary list by summing its elements (note again the one element returned tuple corresponding to a single objective fitness). Subsequently, a `Toolbox` is instantiated in which the necessary operators are registered. The first operator, on line 8, produces binary values, in this case integers in $[0, 1]$, using the standard *random* module. The alias `individual`, on line 9, is assigned to the helper function `initRepeat`, which takes a container as the first argument, a function that generates content as the second argument, and the number of repetitions as the last argument. Thus, calling the individual function instantiates an `Individual` of `n=100` bits by calling repeatedly the registered `attr_bool` function. The same repetition initializer is used on the next line to produce a population as a list of individuals. The missing number of repetitions `n` will be given later in the program. Subsequently, on lines 11 to 14, the evaluation, crossover, mutation and selection operators are registered with all of their parameters.

The main program starts at line 16. First, a population of `n=300` individuals is instantiated. Then, the algorithm, provided with the population and the toolbox, is run for `ngen=40` generations with `cxpb=0.5` probability of mating and `mutpb=0.2` probability of mutating an individual for each generation. Finally, on line 35, the best individual of the resulting population is selected and displayed on screen.

Controlling Everything

When developing, researching or using EAs, pre-implemented canned algorithms seldom do everything that is needed. Usually, developers/researchers/users have to dig into the framework to tune, add or replace a part of the original algorithm. DEAP breaks with the traditional black-box approach on that precise point; it encourages users to rapidly build their own algorithms. With the different tools provided by DEAP, it is possible to design a nimble algorithm that tackles most problems at hand.

Starting from the previous OneMax solution of Figure 1(a), a first decomposition of the algorithm replaces the canned `eaSimple` function (line 17) by the generational loop illustrated in Figure 1(b). Again, this example is exhaustive but still very simple. On the first 3 lines, the evaluation function is applied to every individual in the population by the `map` function contained in every toolbox. Next, on line 18, a loop over both the population and the evaluated fitnesses sets each individual's fitness value. Thereafter, on line 20, the generational loop begins. It starts by selecting `k` individuals from the population. Then, the selected individuals are varied by crossover **and** mutation using the `varAnd` function. A second variation scheme `varOr` can also be used, where the individuals are produced by crossover **or** mutation. Once modified, the individuals are evaluated for the next iteration. Only freshly produced individuals have to be evaluated; they are filtered by their fitness validity; `valid` property of the fitness (line 31). This version of the program provides the possibility to change the stopping criterion and add components to the evolution.

(a)

```
1  import random
2  from deap import algorithms, base, creator, tools

3  creator.create("FitnessMax", base.Fitness, weights=(1.0,))
4  creator.create("Individual", list, fitness=creator.FitnessMax)

5  def evalOneMax(individual):
6      return (sum(individual),)

7  toolbox = base.Toolbox()
8  toolbox.register("attr_bool", random.randint, 0, 1)
9  toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=100)
10 toolbox.register("population", tools.initRepeat, list, toolbox.individual)
11 toolbox.register("evaluate", evalOneMax)
12 toolbox.register("mate", tools.cxTwoPoint)
13 toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
14 toolbox.register("select", tools.selTournament, tournsize=3)

15 if __name__ == "__main__":
16     pop = toolbox.population(n=300)
17     algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=40)
35     print(tools.selBest(pop, k=1)[0])
```

(b)

```
17 fitnesses = toolbox.map(toolbox.evaluate, pop)
18 for ind, fit in zip(pop, fitnesses):
19     ind.fitness.values = fit

20 for g in range(ngen):
21     pop = toolbox.select(pop, k=len(pop))
22     pop = algorithms.varAnd(pop, toolbox, cxpb, mutpb)
31     invalids = [ind for ind in pop if not ind.fitness.valid]
32     fitnesses = toolbox.map(toolbox.evaluate, invalids)
33     for ind, fit in zip(invalids, fitnesses):
34         ind.fitness.values = fit
```

(c)

```
22 offspring = [toolbox.clone(ind) for ind in pop]

23 for child1, child2 in zip(offspring[::2], offspring[1::2]):
24     if random.random() < cxpb:
25         toolbox.mate(child1, child2)
26         del child1.fitness.values, child2.fitness.values

27 for mutant in offspring:
28     if random.random() < mutpb:
29         toolbox.mutate(mutant)
30         del mutant.fitness.values
```

Fig. 1: OneMax example with DEAP. (a) Simpler version relying on the pre-implemented eaSimple algorithm. (b) Unboxing of the eaSimple algorithm to control selection, variation and evaluation. (c) Unfolding of the variation to handle crossover and mutation.

An even greater level of detail can be obtained by substituting the `varAnd` function by its full content, presented in Figure 1(c). This listing starts with the duplication of the population by the `clone` tool available in every toolbox. Then, the crossover is applied to a portion of consecutive individuals. Each modified individual sees its fitness invalidated by the deletion of its values on line 26. Finally, a percentage of the population is mutated and their fitness invalidated. This variant of the algorithm provides control over the application order and the number of operators, among other aspects.

The explicitness in which algorithms are written with DEAP clarifies the experiments. This eliminates any ambiguity on the different aspects of the algorithm that could, when overlooked, jeopardize the reproducibility and interpretation of results.

Genetic Programming

DEAP also includes every component necessary to design genetic programming algorithms with the same ease as for genetic algorithms. For example, the most commonly used tree individual can be created as follows:

```
import math, operator
from deap import base, creator, tools, gp

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", gp.PrimitiveTree,
     fitness=creator.FitnessMin)
```

The primitive tree is provided in the *gp* module since it is one of the few data types the Python standard library does not provide. The primitives and terminals that will populate the trees are regrouped in a primitive set. The following listing presents a primitive set instantiation with basic operators provided by the standard library *operator* module. The `arity` of a primitive is its number of operands.

```
pset = gp.PrimitiveSet(name="MAIN", arity=1)
pset.addPrimitive(operator.add, arity=2)
pset.addPrimitive(operator.sub, arity=2)
pset.addPrimitive(operator.mul, arity=2)
pset.addPrimitive(operator.neg, arity=1)
```

Functions that initialize individuals and populations are registered in a toolbox just as in the preceding genetic algorithm example. DEAP implements the three initialization methods proposed by Koza [4] to generate trees: full, grow, and half-and-half.

```
toolbox = base.Toolbox()
toolbox.register("expr", gp.genFull, pset=pset, min_=1, max_=3)
toolbox.register("individual", tools.initIterate,
        creator.Individual, toolbox.expr)
toolbox.register("population", tools.initRepeat,
        list, toolbox.individual)
```

We may now introduce an example of a symbolic regression evaluation function.

```
def evaluateRegression(individual, points, pset):
    func = gp.compile(expr=individual, pset=pset)
    sqerrors = ((func(x) - (x**4 + x**3 + x**2 + x))**2 for x in points)
    return (math.sqrt(sum(sqerrors) / len(points)),)
```

First, the `gp.compile` function transforms the primitive tree into its executable form, a Python function, using a primitive set `pset` given as the evaluation function's third argument. Then, the rest is simple math: we compute the root mean squared error between the individual's program and the target $x^4+x^3+x^2+x$ on a set of `points`, the evaluation function's second argument.

Next, the evaluation function and the variation operators are registered similarly to Figure 1, while line 14 to the end remain exactly the same. Furthermore, using external libraries such as NetworkX [5] and PyGraphviz [6], the best primitive trees can be visualized[1] as presented in Figure 2.

The primitives are not limited to standard library operators, any function or instance method can be added to a primitive set. Terminals can be any type of objects and even functions without argument. The next example, presented in Figure 3, takes advantage of this flexibility and reduces the runtime of the previous example by vectorizing the evaluation using Numpy [7], a library of high-level mathematical functions operating on multidimensional arrays.

The idea is to evolve a program whose argument is a vector instead of a scalar. Most of the code remains identical, only minor modifications (highlighted in Figure 3) are required. First, we replace the operators in the primitive set by Numpy operators that work on vectors (lines 6 to 9). Then, we remove the loop from the evaluation function (line 12), since it is implicit in the operators. Finally, we replace the `sum` and `sqrt` functions by their faster Numpy equivalent (line 13) and our regression problem is now vectorized.

[1] See the notebook version of this article for the complete code to visualize the tree: `http://github.com/DEAP/notebooks`.

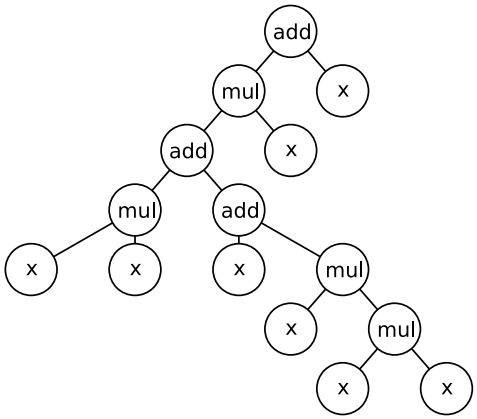


Fig. 2: Example of a GP individual generated with DEAP.

The execution is thereby significantly improved as the scalar example runs in around 3 seconds to optimize the regression on 20 points, while the vectorial runtime is identical but for a regression on 1000 points. By modifying only 6 lines of code, not only are we able to vectorize our problem, but the runtime is reduced by a **factor of 50**.

In addition to the wide support of function and object types, DEAP's *gp* module also supports automatically defined functions (ADF), strongly typed genetic programming (STGP), and object-oriented genetic programming (OOGP), for which examples are provided in the library documentation.

Distributed Island Model

The island model paradigm consists in multiple populations evolving separately and exchanging individuals on a regular basis. The final example illustrates how this scheme can be implemented with DEAP and SCOOP. The code presented in Figure 4 evolves 5 islands of 300 individuals. The algorithm runs for 40 generations, and every 10 generations, the 15 best individuals from one island are migrated to the next, following a ring topology.

```
1  import numpy
2  from deap import algorithms, base, creator, tools, gp

3  creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
4  creator.create("Tree", gp.PrimitiveTree, fitness=creator.FitnessMin)

5  pset = gp.PrimitiveSet(name="MAIN", arity=1)
6  pset.addPrimitive(numpy.add, arity=2)
7  pset.addPrimitive(numpy.subtract, arity=2)
8  pset.addPrimitive(numpy.multiply, arity=2)
9  pset.addPrimitive(numpy.negative, arity=1)

10 def evaluateRegression(individual, points, pset):
11     func = gp.compile(expr=individual, pset=pset)
12     sqerrors = (func(points)-(points**4 + points**3 +
↪                                points**2 + points))**2
13     return (numpy.sqrt(numpy.sum(sqerrors) / len(points)),)

14 toolbox = base.Toolbox()
15 toolbox.register("expr", gp.genFull, pset=pset, min_=1, max_=3)
16 toolbox.register("individual", tools.initIterate, creator.Tree,
↪                  toolbox.expr)
17 toolbox.register("population", tools.initRepeat, list,
↪                  toolbox.individual)
18 toolbox.register("evaluate", evaluateRegression,
↪                  points=numpy.linspace(-1, 1, 1000), pset=pset)
19 toolbox.register("mate", gp.cxOnePoint)
20 toolbox.register("expr_mut", gp.genFull, min_=0, max_=2)
21 toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut,
↪                  pset=pset)
22 toolbox.register("select", tools.selTournament, tournsize=3)

23 if __name__ == "__main__":
24     pop = toolbox.population(n=300)
25     algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=40)
26     print(tools.selBest(pop, k=1)[0])
```

Fig. 3: Vectorized Genetic Programming example.

The for-loop starting at line 6 maps the algorithm to each island (line 7), making them evolve independently for `FREQ=10` generations. Then, the resulting populations are recovered on line 8 and a ring topology migration is applied on line 9, using the built-in `migRing` operator to exchange individuals between islands. Since `eaSimple` uses the operator registered on line 1 to map the evaluation on the individuals, fitnesses are also computed in parallel. Therefore, the computations are distributed at 2 different levels.

The distribution scheme is presented in Figure 5. Each island evolution is executed by a distinct process and so is every evaluation. The listing in Figure 4 is generic. It could thus replace the main section of any of the previously presented examples: lines 16 and 17 in Figure 1(a) and lines 24 and 25 in Figure 3. To run this code on multiple processors, assuming the source code is in file `island.py`, one simply needs to enter the command line: `python -m scoop island.py`

```
1 toolbox.register("map", futures.map)
2 toolbox.register("migrate", tools.migRing, k=15,
↪                  selection=tools.selBest)
3 NGEN, FREQ = 40, 10
4 toolbox.register("algorithm", algorithms.eaSimple, toolbox=toolbox,
↪                  cxpb=0.5, mutpb=0.2, ngen=FREQ, verbose=False)
5 islands = [toolbox.population(n=300) for i in range(5)]
6 for i in range(0, NGEN, FREQ):
7     results = toolbox.map(toolbox.algorithm, islands)
8     islands = [island for island, logbook in results]
9     toolbox.migrate(islands)
```

Fig. 4: Distributed Island Model example.

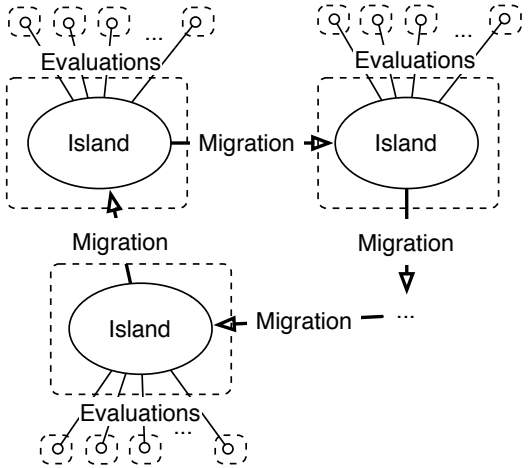


Fig. 5: Island model distribution scheme. Each island runs in a different process and evaluations are also done in parallel. SCOOP takes care of all necessary load balancing.

Evolution Support

DEAP comes with several supporting tools that can be easily integrated into any algorithm. This section presents some of them in the context of the OneMax example (Figure 1).

The first tool, *Statistics*, computes statistics on arbitrary attributes of designated objects, usually the fitness of the individuals. The attribute is specified by a key function at the statistics object instantiation before starting the algorithm, between lines 16 and 17 of Figure 1(a).

```
stats = tools.Statistics(key=operator.attrgetter("fitness.values"))
```

This is followed by the registration of the statistical functions as for a toolbox.

```
stats.register("avg", numpy.mean)
stats.register("min", numpy.min)
stats.register("max", numpy.max)
```

Ultimately, at every generation, a statistical record of the population is compiled using the registered functions.

```
record = stats.compile(pop)
print(record)
```

The statistics compilation produces a dictionary containing the statistical keywords and their respective value. These last lines, added after the evaluation part of Figure 1(b) (line 34), will produce a screen log of the evolution statistics.

For posterity and better readability, statistics can also be logged in a *Logbook*, which is simply a list of recorded dictionaries that can be printed with an elegant layout. For example, the following lines create a new logbook, then record the previously computed statistics and print them to the screen.

```
logbook = tools.Logbook()
logbook.record(gen=g, nevals=300, fitness=record)
print(logbook)
                fitness
            ---------------
gen nevals  avg     min max
0   300     49.9933 35  64
```

The next tool, named *Hall of Fame*, preserves the best individuals that appeared during an evolution. At every generation, it scans the population and saves the individuals in a separate archive that does not interact with the population. If the best solution disappears during the evolution, it will still be available in the hall of fame. The hall of fame can be provided as an argument to the algorithms (Figure 1(a) line 17) as follows:

```
halloffame = tools.HallOfFame(maxsize=10)
algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=40,
                    halloffame=halloffame)
```

Moreover, the hall of fame can be updated manually right after the population is evaluated (Figure 1(b) line 34) with the following line of code.

```
halloffame.update(pop)
```

The hall of fame proposes a list interface where the individuals are sorted in descending order of fitness. Thus, the fittest solution can be retrieved by accessing the list's first element.

```
best = halloffame[0]
```

A Pareto dominance version of the hall of fame is also available. The *Pareto Front* maintains an archive of non-dominated individuals along the evolution. Its interface is the same as the standard hall of fame.

Another tool, called the *History*, tracks the genealogy of the individuals in a population. By wrapping the variation operators, the history saves the parents of each individual. This feature is added to the variation operators of the toolbox with the following lines.

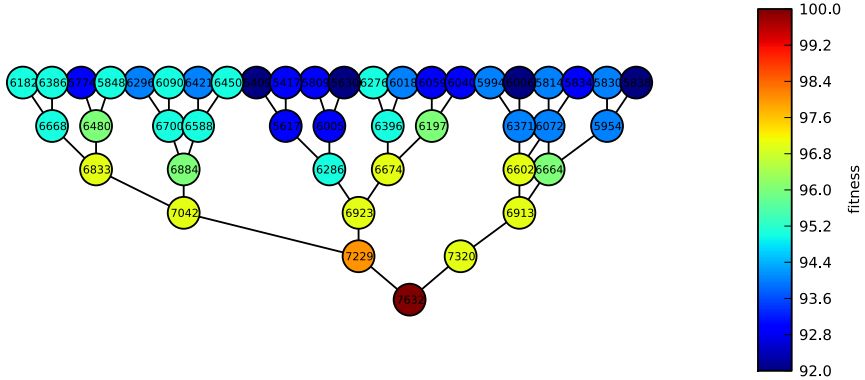


Fig. 6: Genealogy tree of the best individual found in the OneMax problem during the last 5 variation operations. The ancestors are at the top while the fittest offspring is at the bottom. The numbers represent the solution index. A node with two incoming links is the result of a crossover (see 7229), while a node with a single incoming link comes from a mutation (see 7320).

```
history = tools.History()
toolbox.decorate("mate", history.decorator)
toolbox.decorate("mutate", history.decorator)
```

It is therefore possible to determine the genesis of individuals. Figure 6 presents the genealogy of the best individual in the OneMax example for the last 5 variation operations. The graph is produced by the NetworkX library and the following listing.

```
h = history.getGenealogy(haloffame[0], max_depth=5)
graph = networkx.DiGraph(h)
networkx.draw(graph)
```

The last presented tool is a checkpointing facility. Rather than a DEAP object, checkpointing is ensured by the powerful *pickle* standard library module that can serialize almost any Python object. Checkpointing only requires selecting objects that shall be preserved and the write frequency. This is exactly what is done in the following lines that can be added at the end of the generational loop of Figure 1(b).

```
import pickle

if g % freq == 0:
    cp = dict(population=pop, generation=g, rndstate=random.getstate())
    pickle.dump(cp, open("checkpoint.pkl", "w"))
```

These last lines write into a file the population, the generation number, and the random number generator state so that this information can be used later to restart an evolution from this exact point in time. Reloading the data is as simple as reading the pickled dictionary and accessing its attributes.

```
cp = pickle.load(open("checkpoint.pkl", "r"))
pop = cp["population"]
g = cp["generation"]
random.setstate(cp["rndstate"])
```

This simple mechanism provides fault tolerance to any sort of evolutionary algorithms implemented with DEAP. This happens to be critical when exploiting large computational resources where chances of failure grow quickly with the number of computing nodes. Even in very stable execution environments, checkpoints can significantly reduce the amount of time spent experimenting by allowing evolutions to restart and continue beyond the original stopping criteria.

Conclusion

DEAP proposes an agile framework to easily prototype and execute explicit evolutionary algorithms. Its creator module is instrumental for building custom transparent data structures for the problem at hand. Its toolbox gathers all necessary operators and their arguments in a single handy structure. Its design provides straightforward distributed execution with multiple distribution libraries. The presented examples covered only a small part of DEAP's capabilities that include evolution strategies (including CMA-ES), multi-objective optimization (NSGA-II and SPEA-II), co-evolution, particle swarm optimization, as well as many benchmarks (continuous, binary, regression, and moving peaks), and examples (more than 40).

After more than 4 years of development, DEAP version 1.0 has been released in February 2014. DEAP is an open source software, licensed under LGPL, developed primarily at the Computer Vision and Systems Laboratory of Université Laval, Québec, Canada. DEAP is compatible with Python 2 and 3. It has a single dependency on Numpy for computing statistics and running CMA-ES. Try it out and become nimbler too: http://deap.gel.ulaval.ca.

References

[1] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13:2171–2175, 2012.

[2] F.-M. De Rainville, F.-A. Fortin, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: A Python Framework for Evolutionary Algorithms. *In Companion Proceedings of the Genetic and Evolutionary Computation Conference*, pages 85–92, 2012.

[3] Y. Hold-Geoffroy, O. Gagnon, and M. Parizeau. SCOOP: Scalable COncurrent Operations in Python. http://www.pyscoop.org/

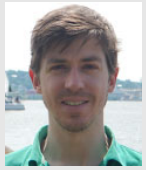
[4] J. R. Koza. Genetic Programming - On the Programming of Computers by Means of Natural Selection. MIT Press, 1992.

[5] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. *In Proceedings of the Python in Science Conference*, pages 11-15, 2008. http://networkx.github.io

[6] A. A. Hagberg, D. A. Schult, and M. Renieris. PyGraphviz a Python interface to the Graphviz graph layout and visualization package. http://networkx.lanl.gov/pygraphviz

[7] E. Jones and T. Oliphant and P. Peterson and others. SciPy: Open source scientific tools for Python. http://www.scipy.org

About the authors



François-Michel De Rainville received his Master's degree from Université Laval, Québec, Canada, in 2010, for his work on interactive designs of experiments to enhance comprehension of complex systems. He is currently pursuing his PhD degree at Université Laval on using a swarm of robots to explore and analyze unknown environments. His major area of interest are robotics, pattern recognition, machine learning, evolutionary algorithms, and computer vision. He is one of the main DEAP developers.

Homepage: http://vision.gel.ulaval.ca/~fmdrainville
Email: francois-michel.de-rainville.1@ulaval.ca

Félix-Antoine Fortin received a M.Sc. in Electrical Engineering from Université Laval in 2010 for his work on automatic surveillance camera placement with genetic algorithms. He is currently completing a Ph.D at Université Laval on multimodal optimization while working as an high performance computing analyst at Calcul Québec, a research consortium for High Performance Computing (HPC). His main research interests are optimization, pattern recognition, evolutionary algorithms, and distributed computing. He is also one of the main developer of DEAP.

Homepage: http://vision.gel.ulaval.ca/~fafortin/
Email: felix-antoine.fortin.1@ulaval.ca

Marc-André Gardner received a B.Sc. in Computer Engineering from Université Laval in 2012. He has worked on bloat control in genetic programming, and is currently completing a M.Sc. in Electrical Engineering on stochastic grammar optimization applied to genetic programming. He has also worked on a task distribution framework in Python, and is a major contributor to DEAP, in addition to being one of its power users.

Email: marc-andre.gardner.1@ulaval.ca

Marc Parizeau is a professor of Computer Engineering at Université Laval, Québec City. He obtained his Ph.D. in 1992 from École Polytechnique de Montréal. His research interests are mainly in the field of intelligent systems, in machine learning for pattern recognition in particular, as well as in parallel and distributed systems. In 2008, he created a High Performance Computing (HPC) center at Université Laval, and is the current scientific Director of Calcul Québec, an HPC consortium for the province of Québec, also one of the four regional divisions of Compute Canada, the national HPC platform.

Homepage: http://vision.gel.ulaval.ca/~parizeau
Email: parizeau@gel.ulaval.ca

Christian Gagné received a B.Ing. in Computer Engineering and a PhD in Electrical Engineering from Université Laval in 2000 and 2005, respectively. He is professor of Computer Engineering at Université Laval since 2008. His research interests are on the engineering of intelligent systems, in particular systems involving machine learning and evolutionary computation. He is member of editorial board of the Genetic Programming and Evolvable Machines journal, and participated to the organization of several conferences. He is also the main developer of Open BEAGLE, a generic C++ framework for evolutionary computation, from which lessons learnt served as inspiration to the design of DEAP.

Homepage: http://vision.gel.ulaval.ca/~cgagne
Email: christian.gagne@gel.ulaval.ca

GECCO-2013 Competitions

Daniele Loiacono

What the Numbers of GECCO-2013 Competitions?

The 2013 GECCO competitions offered six different challenges involving fifteen organizers and overall received the astonishing number of thirty-one submissions. As usually happens at GECCO, competitions represented an opportunity to dig into challenging problems while competing with other members of the GECCO's community. At the same time, they were also a nice spot to showcase some amazing applications of evolutionary computation; the results of the competitions were presented during two lunch-time sessions of the conference. What follows is a brief summary for each one of the six competitions in the program.

Visualizing Evolution

The Visualizing Evolution Competition was organized by David Walker, Richard Everson, and Jonathan Fieldsend. To enter this competition, the participants were asked to exhibit cutting edge visualizations of an evolutionary process. The competition was also connected to the Visualisation Methods in Genetic and Evolutionary Computation workshop where participants were allowed to present their work. The competition attracted two entries and was won by Christian E. Munoz Villalobos, Douglas M. Dias and Marco Aurelio C. Pacheco.

EvoRobocode

The EvoRobocode Competition, organized by Daniele Loiacono and Moshe Sipper, challenged the participants to apply Evolutionary Computation to design a competitive robot tank for the Robocode platform. The winning robot, SCALPbot, was developed by Robin Harper using Grammatical Evolution together with a spatial co-evolution system.

Evolutionary Art, Design and Creativity

The Evolutionary Art, Design and Creativity Competition, organized by Christian Gagné, Amy K. Hoover, Eduardo R. Miranda, and Craig Reynolds, aimed at showcasing human-quality artistic works or creativity enhancing experiences generated by or with the assistance of evolution. Submitted works could be music, images, sculptures, videos, interactive online experiences or any form of expression. The goal was that to exhibit some form of independent creativity through genetic and evolutionary computation. Among several high quality entries, F. Fernández de Vega, L. Navarro, C. Cruz, P. Hernández, T. Gallego and L. Espada won this competition with a study on human creativity from an evolutionary perspective.

GPU for Genetic and Evolutionary Computation

The GPUs for Genetic and Evolutionary Computation Competition, organized by Daniele Loiacono and Antonino Tumeo, focused on the application of genetic and evolutionary computation that can maximally exploit the parallelism provided by low-cost consumer graphical cards. The entries submitted have been evaluated in terms of degree of parallelism obtained, overall speed-up, and programming style. The competition was won by Shigeyoshi Tsutsui and Noriyuki Fujimoto that presented a fast quadratic assignment problem solver based on a multiple GPUs parallelization of an ACO.

Simulated Car Racing

The Simulated Car Racing Competition was organized by Daniele Loiacono and Pier Luca Lanzi. The goal of this competition was to design and submit a controller for a racing car that will compete on a set of unknown tracks. After more than twenty races on three different tracks, the winner was proclaimed: Mr. Racer, the winning entry submitted by Jan Quadflieg, Tim Delbrugger, Kai Verlage and Mike Preuss, combined successfully a CMA-ES to optimize the parameters of the controller with an online learning algorithm to build a model of the track.

The Industrial Challenge

The Industrial Challenge was organized by Martina Friese, Oliver Flasch, Olaf Mersmann, and Thomas Bartz-Beielstein with GreenPocket as industrial partner. Goal of this competition was to develop accurate and efficient forecasting methods and to apply them to real-world smart home time series data. The competition attracted several entries and was won by Farzad Noorian with a solution based on SVMs.

Sponsors

Competitions were sponsored by GreenPocket, that awarded the winner of the Industrial Challenge with an iPad, by NVIDIA, that donated three TESLA K20 GPUs for the best entries, and by GECCO that awarded runner-ups with a small cash prize.

GECCO-2014 “Humies” Awards

July 14, 2014

$10,000 in Awards
The 2014 "Humies" for Human-Competitive Results

Homepage: www.human-competitive.org

Deadline June 2, 2014

Entries are hereby solicited for awards totaling $10,000 for human-competitive results that have been produced by any form of genetic and evolutionary computation (including, but not limited to genetic algorithms, genetic programming, evolution strategies, evolutionary programming, learning classifier systems, grammatical evolution, gene expression programming, differential evolution, etc.) and that have been published in the open literature between the deadline for the previous competition and the deadline for the current competition.

The competition will be held as part of the 2014 Genetic and Evolutionary Computation (GECCO) conference. Presentations of entries will be made at the conference. The awards and prizes will be announced and presented during the conference. See http://www.sigevo.org/gecco-2014/

Important Dates

- Monday June 2, 2014 — Deadline for entries (consisting of one TEXT file and one or more PDF files). Send entries to koza@human-competitive.org
- Monday June 23, 2014 — Finalists will be notified by e-mail
- Thursday July 3, 2014 — Finalists must submit their presentation (e.g., PowerPoint, PDF) for posting on the competition web site. Send presentations to koza@human-competitive.org
- July 12-16,2014 (Sat-Wed) — The GECCO conference
- Monday July 14, 2014 (tentative) — Presentations before judging committee at public session of the GECCO conference
- Wednesday July 16, 2014 (tentative) — Announcement of awards at plenary session of the GECCO conference

If you plan to make an entry for this competition, please check the web site at www.human-competitive.org for updated information prior to submitting your entry. If you make an entry, please re-check this web site periodically prior to the conference for additional (and possible changing) information and instructions.

Judging Committee

- Erik Goodman
- Una-May O’Reilly
- Wolfgang Banzhaf
- Darrell Whitley
- Lee Spector

Call for Entries

Techniques of genetic and evolutionary computation are being increasingly applied to difficult real-world problems — often yielding results that are not merely academically interesting, but competitive with the work done by creative and inventive humans. Starting at the Genetic and Evolutionary Computation Conference (GECCO) in 2004, cash prizes have been awarded for human-competitive results that had been produced by some form of genetic and evolutionary computation in the previous year.

This prize competition is based on published results. The publication may be a paper at the GECCO conference (i.e., regular paper, poster paper, or any other full-length paper), a paper published anywhere in the open literature (e.g., another conference, journal, technical report, thesis, book chapter, book), or a paper in final form that has been unconditionally accepted by a publication and is "in press" (that is, the entry must be identical to something that will be published imminently without any further changes).

The publication may not be an intermediate or draft version that is still subject to change or revision by the authors or editors. The publication must meet the usual standards of a scientific publication in that it must clearly describe a problem, the methods used to address the problem, the results obtained, and sufficient information about how the work was done in order to enable the work described to be replicated by an independent person.

An automatically created result is considered "human-competitive" if it satisfies at least one of the eight criteria below.

(A) The result was patented as an invention in the past, is an improvement over a patented invention, or would qualify today as a patentable new invention.

(B) The result is equal to or better than a result that was accepted as a new scientific result at the time when it was published in a peer-reviewed scientific journal.

(C) The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.

(D) The result is publishable in its own right as a new scientific result independent of the fact that the result was mechanically created.

(E) The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.

(F) The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.

(G) The result solves a problem of indisputable difficulty in its field.

(H) The result holds its own or wins a re gulated competition involving human contestants (in the form of either live human players or human-written computer programs).

Contestants should note that a pervasive thread in most of the above eight criteria is the notion that the result satisfy an "arms length" standard — not a yardstick based on the opinion of the author, the author's own institution (educational or corporate), or the author's own close associates. "Arms length" may be established in numerous ways. For example, if the result is a solution to "a long-standing problem for which there has been a succession of increasingly better human-created solutions," it is clear that the scientific community (not the author, the author's own institution, or the author's close associates) have vetted the significance of the problem. Similarly, a problem's significance may be established if the re sult replicates or improves upon a scientific result published in a peer-reviewed scientific journal, replicates or improves upon a previously patented invention, constitutes a patentable new invention, or replicates or improves a result that was considered an achievement in its field at the time it was first discovered. Similarly, a problem's significance may be established if the result holds its own or wins a regulated competition involving live human players or human-written computer programs. In each of the foregoing examples, the standard for human-competitiveness is being established external to the author, the author's own institution, or the author's close associates. It is also conceivable to rely only on criterion G ("The result solves a problem of indisputable difficulty in its field"); however, if only criterion G is claimed, there must be a clear and convincing argume nt that the problem's "difficulty" is indeed "indisputable."

The competition will be held as part of the annual Genetic and Evolutionary Computation (GECCO) conference. Presentations of entries are to be made at the conference. The awards and prizes will be announced at the conference.

Cash prizes of $5,000 (gold), $3,000 (silver), and bronze (either one prize of $2,000 or two prizes of $1,000) will be awarded for the best entries that satisfy one or more of the criteria for human-competitiveness. The awards will be divided equally among co-authors unless the authors specify a different division at the time of submission.

Prizes are paid by check in U.S. dollars.

Detailed Instructions for Entering the "Humies"

If you plan to make an entry into this competition, please check the web site at www.human-competitive.org for updated information prior to submitting your entry. If you make an entry, please re-check the web site prior to the conference for possible changes in the instructions or the schedule.

All entries are to be sent electronically to koza@human-competitive.org. All entries will be promptly acknowledged, so please make an inquiry if you do not receive a prompt acknowledgment.

An entry must consist of one TEXT file and one or more PDF files. If the same authors are making multiple entries, please submit separate e-mails, each containing both the required TEXT and PDF file(s) supporting the entry.

The TEXT file must contain the fo llowing 10 items. Please be very careful to include ALL required information. Contestants are alerted to the fact that items 6 and 9 are especially important and will be the main basis by which entries will be judged. The papers and presentations from earlier competitions (starting in 2004) are posted at the competition web site at www.human-competitive.org and may be informative.

1. the complete title of one (or more) paper(s) published in the open literature describing the work that the author claims describes a human-competitive result;
2. the name, complete physical mailing address, e-mail address, and phone number of EACH author of EACH paper(s);
3. the name of the corresponding author (i.e., the author to whom notices will be sent concerning the competition);
4. the abstract of the paper(s);
5. a list containing one or more of the eight letters (A, B, C, D, E, F, G, or H) that correspond to the criteria (see above) that the author claims that the work satisfies;
6. a statement stating why the result satisfies the criteria that the contestant claims (see examples of statements of human-competitiveness as a guide to aid in constructing this part of the submission);
7. a full citation of the paper (that is, author names; publication date; name of journal, conference, technical report, thesis, book, or book chapter; name of editors, if applicable, of the journal or edited book; publisher name; publisher city; page numbers, if applicable);
8. a statement either that "any prize money, if any, is to be divided equally among the co-authors" OR a specific percentage breakdown as to how the prize money, if any, is to be divided among the co-authors; and
9. a statement stating why the judges should consider the entry as "best" in comparison to other entries that may also be "human-competitive;"
10. An indication of the general type of genetic or evolutionary computation used, such as GA (genetic algorithms), GP (genetic programming), ES (evolution strategies), EP (evolutionary programming), LCS (learning classifier systems), GE (grammatical evolution), GEP (gene expression programming), DE (differential evolution), etc.

The PDF file(s) are to contain the paper(s). The strongly preferred method is that you send a separate PDF file for each of your paper(s) relating to your entry. Both the text file and the PDF file(s) for each entry will be permanently posted on a web page shortl y after the deadline date for entries (for use by the judges, conference attendees, and anyone else who is interested) and will remain posted on the web as a permanent record of the competition. If your paper is only available on the publisher's web site and your publisher specifically requires that your published paper may appear only on your own personal page, the second choice is that you send link(s) to a separate web page on your web site containing link(s) to the PDF file(s) of the paper(s) that constitute your entry. This separate web page is to contain nothing else, so the interested parties may quickly locate your paper(s). If you use this second-choice option, you must ALSO supply a link to a permanent web site maintained by your publisher where your specific paper may be viewed or purchased (that is, not a link merely to the publisher's general home page, but a link to the specifi c web page containing your paper on the publisher's site). The objective, in each case, is to provide a permanent record of the entries and to make it easy for anyone to locate the entries.

Generally, only one paper should be submitted. Note that this is a competition involving a result that satisfies the criteria for being human-competitive (not a competition involving an author's entire body of work). More than one paper should be submitted only if no one paper fully describes the result or methods.

The judging committee will review all entries and identify a short list of approximately 6–10 finalists for presentation at the GECCO conference. Finalists will be notified by an e-mail to the corresponding author. Please acknowledge receipt of this message, so the judges know that you received your notice. Finalists must then make a short oral presentation to the judging committee at a public session of the GECCO conference. The presentations will be held on one of the early days of the conference and the winners will be announced a day or two later.

Finalists must submit their presentation (e.g., a PowerPoint, PDF) by e-mail to koza at human-competitive dot org. All submissions will be promptly acknowledged, so please make an inquiry if you do not receive a prompt acknowledgment. These presentations will be added to the web page for the competition.

At the GECCO conference, there will be 10-minute oral presentations by the finalists to the judging committee. The presentations will be open to all conference attendees at a special session of the conference. The oral presentation should primarily focus on

- why the result qualifies as being human-competitive and
- why the judges should consider the entry as "best" in comparison to other entries that may also be "human-competitive" since, as previously mentioned, these are the two main standards by which entries will be judged by the judges.

In the short oral presentation to the judges, a description of the work itself is decidedly secondary. By the time of the presentation the judges will be familiar with the papers. Thus, the focus of the presentation is on reasons why the work being presented should win a prize —– not an explanation or presentation of the work itself. In the unlikely event that a presenter is scheduled to make a presentation elsewhere at the GECCO conference at the same time, pleas e notify the judging committee, so they can rearrange time slots. After the oral presentations, the award committee will meet and consider the presentations.

The presenting author for each entry must register for the GECCO conference.

A judge will recuse himself or herself if he or she is closely associated with a finalist (e.g., a current academic advisor, current collaborator, co-author with the finalist of related work).

Additional information is at www.human-competitive.org

Calls and Calendar

April 2014

Evostar 2014 - EuroGP, EvoCOP, EvoBIO and EvoWorkshops
April 23-25, 2014, Granada, Spain
Homepage: www.evostar.org

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.
The five conferences include:

- EuroGP 17th European Conference on Genetic Programming
- EvoBIO 12th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Computational Biology
- EvoCOP 14th European Conference on Evolutionary Computation in Combinatorial Optimisation
- EvoMUSART 3rd International Conference (and 12th European event) on Evolutionary and Biologically Inspired Music, Sound, Art and Design
- EvoApplications 16th European Conference on the Applications of Evolutionary and bio-inspired Computation including the following tracks
  - EvoCOMNET Application of Nature-inspired Techniques for Communication Networks and other Parallel and Distributed Systems
  - EvoCOMPLEX Applications of algorithms and complex systems
  - EvoENERGY Evolutionary Algorithms in Energy Applications
  - EvoFIN Track on Evolutionary Computation in Finance and Economics
  - EvoGAMES Bio-inspired Algorithms in Games
  - EvoHOT Bio-Inspired Heuristics for Design Automation
  - EvoIASP Evolutionary computation in image analysis, signal processing and pattern recognition
  - EvoINDUSTRY The application of Nature-Inspired Techniques in industrial settings
  - EvoNUM Bio-inspired algorithms for continuous parameter optimisation
  - EvoPAR Parallel and distributed Infrastructures
  - EvoRISK Computational Intelligence for Risk Management, Security and Defense Applications
  - EvoROBOT Evolutionary Computation in Robotics
  - EvoSTOC Evolutionary Algorithms in Stochastic and Dynamic Environments

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

July 2014

GECCO 2014 - Genetic and Evolutionary Computation Conference

July 12-16, 2014, Vancouver, BC, Canada

Homepage: http://www.sigevo.org/gecco-2014

Workshop Submission Deadline March 28, 2014

The Genetic and Evolutionary Computation Conference (GECCO-2014) 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.

Workshop Submission Deadlines

Workshop Submission Deadline	March 28, 2014
Decision Notification	April 15, 2014
Camera-ready Submission	April 25, 2014
Conference	July 12-16, 2014

Organizers

General Chair:	Dirk Arnold
Editor-in-Chief:	Christian Igel
Local Chair:	Elena Popovici
Publicity Chair:	Christian Gagné
Tutorials Chair:	Mengjie Zhang
Workshops Chair:	Per Kristian Lehre
Competitions Chairs:	Amy K. Hoover
Business Committee:	Jürgen Branke
	Darrell Whitley
EC in Practice Chairs:	Thomas Bartz-Beielstein
	Anna I Esparcia-Alcazar
	Jörn Mehnen

Venue

Vancouver is a large and multicultural Canadian city with a distinctive West Coast twist. Nestled between the sea and the mountains, it is a vibrant city close to nature. The remarkable Stanley Park as well as the 22 km seawall, which offers unparalleled opportunities for walking, cycling, and rollerblading, are in walking distance from downtown. The diversity of the city's restaurant scene is second to none. Vancouver has regularly been ranked as one of the most livable cities worldwide and is a beautiful destination to visit.

Travelling to Vancouver is easy and convenient, with direct flights from many cities in North America, Europe, Asia, and Oceania, and a rail connection between the airport and downtown (Canada Line). Visit Tourism Vancouver Web site for more information.

The conference will be held at the Sheraton Wall Centre Vancouver. Located in Downtown Vancouver's shopping and entertainment districts, it is within walking distance of the city's best restaurants and most attractions.

More Information

Visit www.sigevo.org/gecco-2014 for information about electronic submission procedures, formatting details, student travel grants, the latest list of tutorials and workshop, late-breaking abstracts, and more.

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

2014 IEEE World Congress on Computational Intelligence

July 6-11, 2014, Beijing, China

Homepage: http://www.ieee-wcci2014.org/

The IEEE World Congress on Computational Intelligence (IEEE WCCI) is the largest technical event in the field of computational intelligence. IEEE WCCI 2014 will host three conferences: The 2014 International Joint Conference on Neural Networks (IJCNN 2014), the 2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2014), and the 2014 IEEE Congress on Evolutionary Computation (IEEE CEC 2014). IEEE WCCI 2014 will engage in cross-fertilization among the three big areas and 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.

IEEE WCCI 2014 will be held in Beijing, the capital of the People's Republic of China. Beijing is the nation's political, economic, and cultural center as well as China's most important center for international trade and communications. With the biggest central square in the world - Tiananmen Square, the Forbidden City - the largest and best-preserved imperial palace complex, a superbly preserved section of the Great Wall, as well as the largest sacrificial complex in the world - the Temple of Heaven, Beijing attracts both domestic and foreign visitors who all come to wonder at its long history and unique cultural relics.

For more information visit http://www.ieee-wcci2014.org

August 2014

IEEE Conference on Computational Intelligence and Games (CIG-2014)

August 26 - 29, 2014, Dortmund, Germany

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

Submission deadline: April 1, 2014

Conference: August 26 - 29, 2014

Games can be used as a challenging scenery for benchmarking methods from computational intelligence since they provide dynamic and competitive elements that are germane to real-world problems. This conference brings together leading researchers and practitioners from academia and industry to discuss recent advances and explore future directions in this field.

The IEEE Conference on Computational Intelligence and Games is the premier annual event for researchers applying computational and artificial intelligence techniques to games. The domain of the conference includes all sorts of CI/AI applied to all sorts of games, including board games, video games and mathematical games.

The yearly event series started in 2005 as symposium, and is a conference since 2009. An overview over the past CIG conferences is available at hrefhttp://www.ieee-cig.orgwww.ieee-cig.org, where you also find the proceedings. CIG 2014 will be hosted in the Park Inn hotel in the city center of Dortmund, a vibrant, technology-oriented city in the Ruhr area, Germany's largest metropolitan area with around 5 million people.

Topics of interest include, but are not limited to:

- 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

- Learning in games
- Procedural content generation
- Player/opponent modeling in games
- Player affective modeling
- Player satisfaction and experience in games
- Computational and articial intelligence based game design
- Intelligent interactive narrative
- Theoretical or experimental analysis of CI techniques for games
- Non-player characters in games
- Comparative studies and game-based benchmarking
- Applications of game theory

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.

General Chairs

Günter Rudolph, TU Dortmund, Germany
Mike Preuss, WWU Münster, Germany

Program Chairs

Mirjam Eladhari, University of Malta
Moshe Sipper, Ben-Gurion University of the Negev, Israel

Tutorials/Special Sessions Chair

Philip Hingston, Edith Cowan University, Perth, Australia

Competition Chair

Simon Lucas, University of Essex, UK

Keynote Chair

Gillian Smith, Northeastern University, Boston, USA

Proceedings Chair

Paolo Burelli, Aalborg University, Copenhagen, Denmark

Important Dates

Special session proposals:	March 1, 2014
Tutorial proposals:	April 1, 2014
Paper submission:	April 1, 2014
Conference:	August 26-29, 2014

For more information please visit: http://www.cig2014.de

September 2014

PPSN 2014 – International Conference on Parallel Problem Solving From Nature
September 13-17, 2014, Ljubljana, Slovenia
Homepage: http://ppsn2014.ijs.si
Deadline: March 17, 2014

The 13th International Conference on Parallel Problem Solving from Nature (PPSN XIII) will be organized by the Jožef Stefan Institute, Ljubljana, Slovenia, and held at the Ljubljana Exhibition and Convention Centre on September 13-17, 2014. The conference aims to bring together researchers and practitioners in the field of Natural Computing. Natural Computing is the study of computational systems that 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.

Paper Presentation Following the well-established tradition of PPSN conferences, all accepted papers will be presented during poster sessions. Each session will contain several papers, 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 deeper understanding of the papers. All accepted papers will be published in the proceedings as a volume of the Lecture Notes in Computer Science (LNCS) Springer series. The format should follow the LNCS style (http://www.springer.de/comp/lncs/authors.html). Prospective authors are invited to contribute their high-quality original results in the field of Natural Computing as papers of no more than 10 pages.

General Chair

Bogdan Filipič, Jožef Stefan Institute, Slovenia

Honorary Chair

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

Program Co-Chairs

Thomas Bartz-Beielstein, Cologne University of Applied Sciences, Germany
Jürgen Branke, University of Warwick, UK
Jim Smith, University of the West of England, UK

Tutorials Chairs

Shih-Hsi "Alex" Liu, California State University, Fresno, USA
Marjan Mernik, University of Maribor, Slovenia

Workshop Chairs

Evert Haasdijk, VU University Amsterdam, The Netherlands
Tea Tušar, Jožef Stefan Institute, Slovenia

Publication Chair

Jurij Šilc, Jožef Stefan Institute, Slovenia

Local Organizer

Gregor Papa, Jožef Stefan Institute, Slovenia

Important dates

Paper submission	March 17, 2014
Author notification	May 19, 2014
Camera-ready paper submission	June 2, 2014
Early registration	June 4, 2014
Conferencee	September 13-17, 2014

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 to 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.