Assignment Proposals from Every Mentor

To share with students (alphabetical order)

Anne Auger

Summary: Empirical experimentation is at the core of many scientific studies in Evolutionary Computation. For instance to assess the performance of an optimizer or to compare its performance to the one of other optimizers, to test a scientific hypothesis … There are typically three tasks to conduct: 1) design the experimental protocol, 2) program (and run) the numerical experiments and the postprocessing to display the results and 3) interpret critically the results.

The second step is typically the most time-consuming but yet the easiest while 1) and 3) are more difficult and often overlooked.

In the assignments (A) and (B), you will typically have to focus on step 3) while you will be using the COmparing Continuous Optimizers (COCO) platform (https://github.com/numbbo/coco) and the already benchmarked datasets available (http://coco.gforge.inria.fr/).

(A) Benchmark the performance of a continuous optimizer of your choice (ask the mentor for guidance on one if you do not know which one to choose) by using the COCO platform (follow the getting started section of https://github.com/numbbo/coco).
Understand the meaning of the different graphs displaying the results. Carefully investigate the quantitative performance, interpret the performance observed (carefully look at the performance per function, per function class, per type of difficulty). Is it surprising with respect to what you know about the optimizers?
Write a small report.

(B) Compare the performance of three (or more) optimizers of your choice from the already benchmarked methods available. Download the datasets available (http://coco.gforge.inria.fr/doku.php?id=algorithms), postprocess the data using the COCO platform as in (A). Look and interpret quantitatively at the results. (the quantitative aspect is important, what we typically do not want “Algorithm X is better than algorithm Y on 10 functions”, rather “Algorithm X is more than 100 times faster than algorithm Y on this type of functions” …).
Motivate your *a priori* choice of optimizers and write about the performance assessment.

(C) Find out about the dependency on the condition number of the convergence rate of the (1+1)-ES with one-fifth success rule.
Investigate for the (1+1)-ES with one-fifth success rule without covariance matrix adaptation the dependency of the convergence rate in the condition number. Define an experimental protocol (to do so, look at the reference [1]) and implement it. Interpret the results.

[1] Auger, A., Hansen, N., Perez Zerpa, J., Ros, R. & Schoenauer, M. (2009), **Experimental Comparisons of Derivative Free Optimization Algorithms**, In 8th International Symposium on Experimental Algorithms. Dortmund (5526), pp. 3-15. Springer Verlag.

Expected output and deliverable: For (A) and (B): learn to critically do a performance assessment study (particularly look at quantitative aspect). For (C): learn to conduct a sound experimental study.
Deliverable: small report and presentation.

Juergen Branke

Summary: The assignments I propose all focus on proper empirical comparisons of evolutionary algorithms.
Assignment 1: A proper statistical analysis is key for empirical research. Go through the papers at GECCO, compile some summary statistics regarding what statistical tests are used. Talk to some of the presenters and ask why they have or have not used a particular test. Compile a recommendation which test should be used depending on the experiments performed.

Assignment 2: Take two standard EAs implemented in any package, or even a single algorithm, but with two different parameter settings. Compare them on an optimisation problem of your choice using a statistical test. How does the number of replications influence the results? How does the

runtime (number of evaluations) influence your conclusions? What do your results say about the relative performance on some other optimisation problem? On the same optimisation problem but a larger number of dimensions?

Assignment 3: Choose any paper presented at GECCO, and try to replicate its results.

Expected output and deliverable:

- Short written report, summarizing the main observations and conclusions
- 10 min presentation to be held in front of the panel

Carlos Coello

Summary: Implement a version of the S Metric Selection Evolutionary Multiobjective Optimization Algorithm (SMS-EMOA) in which the density estimator is one of the following performance indicators (each indicator corresponds to a different assignment):

1) Delta_p,

2) R2

3) IGD+

Expected output and deliverable: A report and the source code (in the programming language of your choice) of this algorithm. A basic validation of this implementation must include at least 3 test problems (e.g., DTLZ1, DTLZ2 and DTLZ4) and 3 performance indicators (hypervolume, inverted generational distance and spacing). Compare results with respect to NSGA-II.

Carola Doerr

Summary and expected deliverables: **Adaptive Operator Selection**

Each assignment has two parts.

A) Literature Survey:

Summarize and compare the principal approaches to control the parameters of an EA proposed in the following four papers (I can send them to you by e-mail if you do not have access to them).

- Álvaro Fialho, Luís Da Costa, Marc Schoenauer, Michèle Sebag: Analyzing bandit-based adaptive operator selection mechanisms. Ann. Math. Artif. Intell. 60(1-2): 25-64 (2010)

- Dirk Thierens: An adaptive pursuit strategy for allocating operator probabilities. GECCO 2005: 1539-1546

- Benjamin Doerr, Carola Doerr, Jing Yang: Optimal Parameter Choices via Precise Black-Box Analysis. GECCO 2016: 1123-1130

- Benjamin Doerr, Carola Doerr: Optimal Parameter Choices Through Self-Adjustment: Applying the 1/5-th Rule in Discrete Settings. GECCO 2015: 1335-1342

(Feel free to add other adaptation mechanisms to this comparison, if you find another one that catches your interest.)

B) Empirical Study:

Chose a benchmark problem and an algorithmic framework that you find interesting and test the different parameter control mechanisms. Report your findings in a suitable way (i.e., using plots and text to describe performance for different problem dimensions and/or fitness evolution over time). How do they compare to static parameter settings? Please also describe the difficulties in implementing the different strategies.

Examples for benchmark problems that you could take a look at (you can also use continuous benchmark problems, if you prefer):

- "toy problems" like OneMax, LeadingOnes, linear functions, etc.
- combinatorial problems like MaxSAT, Minimum Spanning Trees, etc.
- existing benchmark suits

Examples for algorithmic frameworks:

- simple stochastic search like (1+1) EA with adaptive mutation strengths, local search with varying neighborhood size, etc
- population-based methods
- swarm algorithms
- etc.

Manuel López-Ibáñez

Summary: *A Survey on Pitfalls and Maxims in the Empirical Analysis of Algorithms*

The empirical analysis of algorithms, in particular stochastic algorithms such as evolutionary algorithms (EAs) and other metaheuristics, is challenging because of differences in implementations, simulation environments, human biases, and a lack of standard guidelines that are fully accepted and implemented by the community.

There are two main categories of experimental algorithmic research: (1) experiments aimed exclusively at determining the best method among several (i.e., a horse-race) and (2) experiments that try to identify and explain the causes of the observed results [5]. Ideally, even papers in the first category should also include the latter type of experiments [9]. In either case, common pitfalls are extracting conclusions on the basis of a few (sometimes easily solvable) instances or few runs [4, 7]; lack of statistical testing even to the extreme of not reporting the variance of results [4,5]; the use of default parameter settings for algorithms that were designed for different problems or experimental conditions; not reporting exactly how algorithms were tuned [6, 8]; using the same instances for tuning (or preliminary experiments) as later used for comparing and evaluating algorithms leading to over-tuning [2]; focusing too much on the best solution out of N runs (or on the fact that the best-known solution is improved) since these are biased statistics [7]; using CPU-time as a metric or stopping criterion without reporting full CPU (speed, model, cache size, etc.) and implementation language details, etc. A whole range of pitfalls fall within the context of unfair comparisons such as: comparing two methods in terms of iterations/generations where each iteration implies different number of solution evaluations or computation time, measuring CPU-time for algorithms implemented in completely different languages or run on significantly different machines, comparing algorithms in terms of the best solution found, arbitrarily selecting a subset of a benchmark set and discarding “too-easy” or “too-hard” problem instances [3], etc. Good practices include reporting negative results [4], ensuring reproducibility by providing source code and/or raw data [4, 7], identifying sources of variation and performing statistical analysis [1], presenting results graphically taking into account both average behaviour and variance [7], etc.

Expected output and deliverable: The goal of this assignment is to survey the papers and posters presented during GECCO 2017 and note what maxims are followed and what potential pitfalls
are the most common [4,5,7]. The survey should be *observational and respectful*, i.e., students should try to learn any details regarding the experimental analysis by attending the presentation/poster, reading the original paper and, only as a last resort and if given the opportunity, by **respectfully** asking the presenters/authors, not from a critical perspective, but purely informational. *The goal of the assignment is NOT to criticize individual papers but to give a general*
overview of what are most common practices and demonstrate self-reflection. The survey should attempt to understand the motivation and goals of the experimental analysis and cover both good and potentially problematic practices, providing suggestions for addressing the latter. The deliverable will be a fully referenced report (3-5 pages, excluding references) grouping papers according to the type of experimental analysis, good and problematic practices observed, a brief commentary on them including a reflection on why problematic practices arise [4], and final

conclusions. Each student should work independently and try to avoid selecting the same presentation/poster as other students.

[1] Barr, R.S., Golden, B.L., Kelly, J.P., Resende, M.G.C., Stewart, W.R.: Designing and reporting on computational experiments with heuristic methods. J. Heuristics 1(1), 9–32 (1995)
[2] Birattari, M.: Tuning Metaheuristics: A Machine Learning Perspective, Studies in Computational Intelligence, vol. 197. Springer (2009)
[3] Fischetti, M., Monaci, M.: Exploiting erraticism in search. Operations Research 62(1), 114–122 (2014)
[4] Gent, I.P., Grant, S.A., MacIntyre, E., Prosser, P., Shaw, P., Smith, B.M., Walsh, T.: How not to do it. Tech. Rep. 97.27, School of Computer Studies, University of Leeds (1997)
[5] Hooker, J.N.: Testing heuristics: We have it all wrong. J. Heuristics 1(1), 33–42 (1996)
[6] Hoos, H.H.: Programming by optimization. Commun. ACM 55(2), 70–80 (2012)
[7] Johnson, D.S.: A theoretician's guide to the experimental analysis of algorithms. In: Goldwasser, M.H., et al. (eds.) Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, pp. 215–250. American Mathematical Society (2002)
[8] López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3, 43–58 (2016)
[9] López-Ibáñez, M., Prasad, T.D., Paechter, B.: Representations and evolutionary operators for the scheduling of pump operations in water distribution networks. Evol. Comput. 19(3), 429–467 (2011)

John McCall

Summary:
A fundamental question in metaheuristics is "which algorithm should I select to best solve my problem?" Traditionally this question is answered empirically by competitively running particular algorithms, and myriad configurations of algorithms, on a problem instance, or set of instances. More sophisticated approaches use principled methods to search potential algorithm configurations and sets of problem instances with known properties, for example landscape difficulty metrics, are used to provide robust evaluations of variation in performance.

The fact remains however that there are infinitely many problems and infinitely many algorithms to explore empirically, so this task can never be completed. This raises the question of whether we might classify problems, and / or algorithms, ideally in such a way that all algorithms belonging to the same algorithm class have similar performance on all problems belonging to the same problem class.

The three selected papers explore this topic in different ways: monotonicity-invariant function classes [1]; structural coherence of problem and algorithm [2]; and generation of easy and hard hillclimbing problems using fitness interpolation [3]. Each assignment is related to the ideas explored in one or more of them.

Assignment 1: Algorithm classes: Review different approaches used by the community to classify algorithms as represented at GECCO. Compare and contrast two or more algorithm classes of your choice. Do algorithms in each class perform consistently over monotonicity-invariant function classes?

Assignment 2: Problem classes. What does the community mean by problem classes? Review different approaches to defining problem classes as evidenced at GECCO. To what extent do those problem classes have features or structure that are relevant to algorithm performance?

Assignment 3: Programming exercise: adapt the problem generation method used in [3] to a different representation, for example continuous problems. Determine whether or not a similar pattern of easy and hard hillclimbing problems can be generated.

Expected output and deliverable:
A short assignment report and a presentation

Papers:

[1] L. Christie, J. Mccall and D. Lonie, Minimal Walsh Structure and Ordinal Linkage of Monotonicity-Invariant Function Classes on Bit Strings, Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation pp 333 - 340, 2014, ACM

[2] A. Brownlee, J. McCall and L. Christie, Structural coherence of problem and algorithm: An analysis for EDAs on all 2-bit and 3-bit problems, IEEE Congress on Evolutionary Computation (CEC), pp 2066 - 2073, 2015, IEEE

[3] MCCALL, J. A. W., CHRISTIE, L. A. and BROWNLEE, A. E. I, 2015. Generating easy and hard problems using the Proximate Optimality Principle. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15). New York: ACM. pp. 767-768.

Justyna Petke

Summary: Genetic Improvement is the application of evolutionary and search-based optimisation methods to the improvement of existing software. For example, it may be used to automate the process of bug-fixing or execution time optimisation.

Assignment 1: Literature survey: Summarise and present the key developments in the field of automated program repair using genetic programming. In your report try to answer the following questions: How did the initial framework look like? How was it extended to handle large real-world cases? Is it industry ready? The following papers should help you with the review (feel free to include others):

[1] Andrea Arcuri and Xin Yao, "A novel co-evolutionary approach to automatic software bug fixing". CEC 2008, pp. 162-168

[2] Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, Westley Weimer: “A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each”. ICSE 2012: 3-13

[3] Saemundur Oskar Haraldsson, John R. Woodward, Alexander E.I. Brownlee, and Kristin Siggeirsdottir. “Fixing Bugs in Your Sleep: How Genetic Improvement Became an Overnight Success”. GI workshop at GECCO 2017

Assignment 2: Coding exercise: Download and run the GI framework used in the following paper: Justyna Petke, Mark Harman, William B. Langdon and Westley Weimer. “Using Genetic Improvement & Code Transplants to Specialise a C++ Program to a Problem Class”. EuroGP 2014: 137-149. It is available from: www.cs.ucl.ac.uk/staff/J.Petke/papers/gi.zip Please report on your experience. In particular, try to answer the following question: What are the main components of the framework? Please describe and comment on the improvements obtained with the framework.

Assignment 3: Choose a paper presented at GECCO. Present the work and explain how would you extend it. For example, if it’s a tool paper, show how to run it and propose how to extend it or apply to a new application (perhaps write a prototype if appropriate). If it’s a research paper, for example, either state what are the limitations and how would you overcome them or propose an extension that would yield to a new or /wider application of the approach or what are the milestones that need to be overcome to make the approach industry-ready.

Expected output and deliverable: A short report for assignments 1 and 2 and presentation for all.

Mike Preuss

Summary: (all assignments have to do with the microRTS realtime strategy game environment of Santiago Ontanon, to be found at: https://github.com/santiontanon/microrts/wiki)

1) experimentally analyse the importance of specific maps and map sizes (for different existing AIs) with the task of recommending a specific AI for a specific map

2) design, implement and experimentally assess (by doing simulations with different existing AIs) a

new game state evaluation function

3) build a (not too complex) new AI for microRTS on base of one or multiple of the existing ones and test it experimentally against existing AIs

(collaboration of the students may be helpful, even if they have different tasks)

Expected output and deliverable:

- Short written report, summarizing the main ideas, design decisions, experimental results, code concepts, whatever applies
- 10 min presentation to be held in front of the panel

Thomas Stützle

Summary: Traditionally, algorithm development has been a manual, error-prone process. Let's take the example of specific metaheuristics such as "XYZ" (replace XYZ for any metaheuristic you know or want to study such as evolutionary algorithms). Initially, some proposal of a basic version has been made and successively)oftne in a long series of papers) various improvements have been proposed or tested on specific problems. The availability of automated algorithm configuration techniques such as irace, ParamILS, and SMAC makes possible different approaches based on the development of automatically configurable algorithm frameworks that can be configured for specific application problems or application contexts.

1) Literature review: Read the three articles by Hoos, Marmion et al., and Bezerra et al. and summarize these with the perspective on building automatically configurable algorithm frameworks (more precision is given in the presentation).
2) For your favorite "metaheuristic", design a generalized, automatically configurable algorithm framework that can be used to (i) instantiate (all) the well-known specific algorithms belonging to the metaheuristic, (ii) consider how new variants may be generated, and (iii) include novel algorithmic components that have not been considered in the context of the specific metaheuristic.
3) Voluntary, additional task: Starts a prototype implementation of such a framework.

Expected output and deliverable: Written report about the design of the framework, the algorithmic components that are considered, how these are combined, how known algorithm variants can be instantiated from it, how new algorithm variants can be generated. Design of a prototype implementation if task 3 has been started.

Una-May O'Reilly

Summary: I offer 3 assignments that will lead you to focus on specifics topic related to Genetic Programming (GP). In particular, behavioral GP, grammatical evolution and co-evolution. The two (or three) assignments have different options depending if you are at a novice, intermediate or advanced level of knowledge with evolutionary algorithms and genetic programming. I'm hopeful 3 students can identify 3 different options between 3 assignments.

Assignment A: Behavioral Genetic Programming (BGP): This is a means of using subprogram behavioral information to inform evolutionary selection and variation.

Part 1:

Novice/Intermediate/Advanced: Read "Behavioral Programming: A Broader and More Detailed Take on Semantic GP". Krzysztof Krawiec, Una-May O'Reilly. GECCO 2014, pp 935-94.

Part 2:

Novice/Intermediate/Advanced: Investigate the open source code for BGP in Java, i.e. run it and

poke around the source code. I will provide URL to a github repository and a GPTP 2017 paper on an extension to BGP.

Part 3:

Novice: Provide a tutorial and set of external documentation for the software project. Add BGP to the FlexGP website.

Intermediate: Add 1 or more new GP problems to the BGP code base and request a github pull to have your source code become part of the project.

Advanced: Develop a new way to exploit the subprogram fitness information BGP provides.

Assignment B: Coevolution and GP: Coevolution is surprisingly complex. Extensive efforts have been expended to characterize it and they have helpfully provided better clarity. It appears in tandem with GP in the context of coevolving test cases to test programs. There's a number of contexts for this approach.

Part 1: Novice/Intermediate/Advanced: Read Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. 2012. Coevolutionary principles. In Handbook of Natural Computing. Springer, 987–1033. (I can provide a pdf of this if you can't obtain it)

Part 2: Novice/Intermediate/Advanced: Investigate the latest open source code release of PonyGP which has a bare-bones integration of coevolution, i.e. run it and poke around the source code. This code is written in Python. URL to come.

Part 3: Intermediate/Advanced: Extend PonyGP+Coev with some enhancement. E.g., either improve its current example or provide another example. Depending on your coding expertise and knowledge of GP and coevolution, set an appropriately challenging goal. See Andrea Arcuri and Xin Yao. 2007. "Coevolving programs and unit tests from their specification." In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering. ACM, 397–400 for an example if you're stumped for ideas.

Assignment C: Distributed GP. All GP experimental systems take too long to run, this is the inevitable price of being hungry to investigate such an interesting topic! Seriously, we seem to age in place as a GP system executes. One remedy is parallelization. Threaded, GPU (shared memory) and cloud implementations can be challenging to develop when distribution/parallelsim/scalling is NOT the central part of a student's thesis, but, the eariler a system is made system scalable, the less frustrating it will be in terms of waiting for runs to finish.

Part 1: Identify a paper or web-documented project on a distributed GP system that has software associated with it.

One example is FCUBE. Paper is **Bring Your Own Learner! A cloud-based, data-parallel commons for machine learning**. Ignacio Arnaldo, Kalyan Veeramachaneni, Andrew Song, Una-May O'Reilly. Special Issue on Computational Intelligence for Cloud Computing, IEEE Comp. Int. Mag. 10(1): 20-32 (2015) and to look at the software start at: http://flexgp.github.io/FCUBE/.

Or, as part of the FlexGP project, there is a series of different learners that can be parallelized with the FlexGP framework:
FlexGP project website: http://flexgp.csail.mit.edu/
FlexGP learners website: http://flexgp.github.io/gp-learners/
Examples of usage with datasets from the UCI repo: http://flexgp.github.io/gp-learners/blog.html

Two of the learners, "Symbolic regression" and "GPFunction classifier" have GPU capability. Note that the latter, GPFunction, is the one described in Arnaldo I., Veeramachaneni K., O'Reilly UM. (2014) Flash: A GP-GPU Ensemble Learning System for Handling Large Datasets. In: Nicolau M. et al. (eds) Genetic Programming. EuroGP 2014. Lecture Notes in Computer Science, vol 8599. Springer, Berlin, Heidelberg, available at https://link.springer.com/chapter/10.1007/978-3-662-44303-3_2

Symbolic regression website: http://flexgp.github.io/gp-learners/sr.html
GPFunction classifier website : http://flexgp.github.io/gp-learners/gpfunction.html

Their code can be found under the same github repo: https://github.com/flexgp/gp-learners

Part 2 (optional): Try to run the software, i.e. run it and poke around the source code. This may require you to have extensive cloud scaling experience on OpenStack or AWS. Or you may need CUDA experience. If you don't this is likely not a good assignment to choose.

Part 3. Design and plan how you would investigate an open GP problem with the system (i.e. integrate a new research investigation with it, not the one the original authors used to demo it.) Consider how specific needs interact with the desire for libraries/frameworks to be general.

Expected output and deliverable:
I. Presentation material: 3 slides on the paper, 5 slides on the source code covering codebase description, your appraisal and list of MUST-Do's and WOULD-BE-NICE remarks. 10 minutes to present this II. **For coding in part 3**: a demo shown to Una-May (and other mentors possibly).