Advanced Evolutionary Computation (Zaawansowane obliczenia ewolucyjne)

Selected topics and applications
(Wybrane zagadnienia i zastosowania)

Krzysztof Krawiec

Laboratory of Intelligent Decision Support Systems
Institute of Computing Science, Poznan University of Technology, Poznań, Poland
http://www.cs.put.poznan.pl/kkrawiec/

May 26, 2012

Introduction

Course outline

Outline

1. Short introduction to evolutionary computation
2. Part I: Estimation of distribution algorithms
3. Part II: Coevolutionary algorithms
4. Part III: Genetic programming and its variants.

This all interlaced with:

1. Case studies
2. A little bit of theory

Logistics and policy

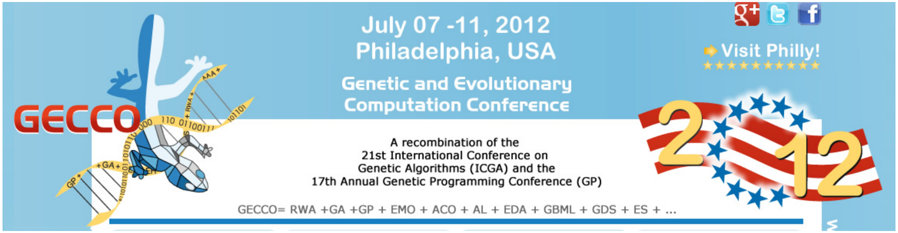
- Yes, the materials are (will be) available online, at:
  - `http://www.cs.put.poznan.pl/kkrawiec/wiki/?n=Zajecia.ZOE`
- Laptops should only be used for taking notes.
- *If anyone uses a laptop or other electronic device for anything not related to class, we will instate a no laptop/no technology policy.*
- Thinking allowed (even thinking aloud). Questions too.
- Freshly made, forgive possible slips.

Background

What is evolutionary computation (EC)?

- A branch of Computational Intelligence (CI) devoted to solving optimization, learning, and design problems using bio-insipred methods, mostly those based on neo-Darwinian evolution.
- CI$\neq$AI
  - CI emphasizes intelligence as an *emergent* phenomenon.
  - CI assumes minimal input of domain knowledge from system's designer.
  - Three main branches: EC, Soft computing (Fuzzy Sets etc.), Neural Networks.
- Offers unconstrained, black-box optimization.
- Successfully applied in many contexts.

A 'formula' for Evolutionary Computation



A recombination of the
21st International Conference on
Genetic Algorithms (ICGA) and the
17th Annual Genetic Programming Conference (GP)

GECCO= RWA +GA +GP + EMO + ACO + AL + EDA + GBML + GDS + ES + ...

- Real-World Applications
- Genetic Algorithms
- Genetic Programming
- Evolutionary Multiobjective Optimization
- Ant Colony Optimization
- Artificial Life
- Estimation of Distribution Algorithms
- Genetic-Based Machine Learning
- Generative and Developmental Systems
- Evolutionary Strategies
- ...

Cont'd

Some variants/applications of EC missing from the equation:

- Evolutionary programming
- Evolutionary neural networks
- Differential evolution
- Search-based software engineering
- ...

Other bio-inspired algorithms

Swarm intelligence

- Ant colony optimization
- Particle swarm optimization
- Bees algorithm
- Cuckoo search

and in a lesser extent also:

- Artificial life (also see digital organism)
- Artificial immune systems
- Cultural algorithms
- Firefly algorithm
- Harmony search
- Learning classifier systems
- Learnable Evolution Model
- Parallel simulated annealing
- Self-organization such as self-organizing maps, competitive learning
- Self-Organizing Migrating Genetic Algorithm
- Swarm-based computing
- Teaching-learning-based optimization (TLBO)

This course

This course covers three, not particularly related to each other, branches of EC:

- Estimation of distribution algorithms (EDA)
- Coevolutionary algorithms (CA)
- Genetic programming (GP)

Evolutionary Computation

Evolutionary Algorithm

- Heuristic bio-inspired global search algorithms
- Operate on populations of candidate solutions
- Candidate solutions are encoded as *genotypes*
- Genotypes get decoded into *phenotypes* when evaluated by the *fitness function* f being optimized.

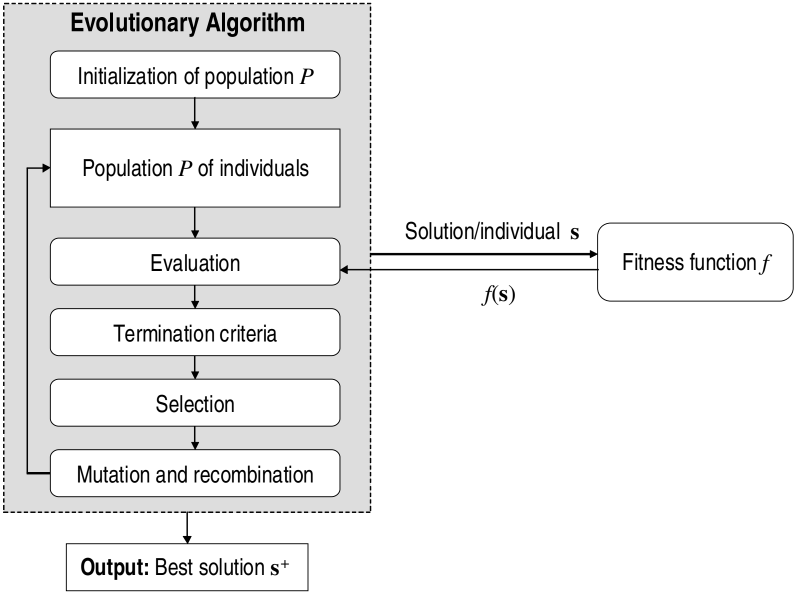
Formulation:

$$p^* = \arg\max_{p \in S} f(p)$$

where

- S is the considered space (*search space*) of *candidate solutions* (*solutions* for short)
- f is a (maximized) fitness function
- p^* is an *optimal solution* (an *ideal*) that maximizes f.

Generic evolutionary algorithm


Evolutionary Algorithm
Initialization of population P
Population P of individuals
Evaluation
Termination criteria
Selection
Mutation and recombination
Solution/individual $\mathbf{s}$
$f(\mathbf{s})$
Fitness function f
Output: Best solution $\mathbf{s}^+$

Important features of EC

- **Iterative**, which implies that:
  - The problem is too difficult to be solved in a single iteration.
  - The search algorithm, while solving the problem, gradually acquires some *knowledge* about it.
- Population-based (**parallel**)
- **Stochastic** (both initialization and execution)
- **Heuristic** (not exact); in most cases not even approximative.

So far it looks as a stochastic, parallel local search. Is there anything new to this?

- Importance of **recombination** (crossover): a recombination operator that makes the solutions exchange certain elements (variable values, features)
- EC performs **global** search

Features of problems

Features of problems that can be tackled using EC

- Black-box optimization
  - how f depends on the independent variables does not have to be known or meet any criteria.
- Variables do not have to be explicitly defined
- Better a good solution today than a perfect tomorrow.
  - Fining an optimum cannot be guaranteed, but in practice a well-performing suboptimal solution is often satisfactory.

Part I: Estimation of Distribution Algorithms

Estimation of Distribution Algorithms

- Also known as:
  - Probabilistic Model-Building Genetic Algorithms, PMBGA (Pelikan 1997)
  - Iterated density estimation algorithms, IDEA (Bosman& Thierens, 2000)
- A class of bio-inspired algorithms that use tools for modeling and sampling of multidimensional probability distribution.

Motivation

What is the purpose of population in EC?

- Serves as a ‘memory’ of the algorithm.
- Defines the current state of the search process.
- Captures certain aspects of the knowledge about the problem, gathered by the search algorithm during the search process.
  - In particular, which combinations of solution components (e.g., variables) are profitable.

Desired properties of population contents:

- Should be diversified (more or less, depending on the stage of the search process).
- Should make generation of new, novel solutions easy.

Estimation of Distribution Algorithms

- In EC, population is a *sample*, which is used for sampling (to produce new samples):
    - entire new populations (*generational* EC),
    - single individuals (*steady-state* EC).
- Can we capture the properties of a sample in a more elegant way?
- The answer: probability distribution.

Estimation of Distribution Algorithms

Key idea: EDAs replace population with a probability distribution (PD).

- PD is used to **sample (draw)** new individuals.
- The information gathered during search is used to **build** (or update) the PD.
- PD changes as the search proceeds.

Notes:

- EDA replaces typical genetic operators (crossover, mutation) wit **building (learning)** and **sampling** a probabilistic model.
- Technically, EDAs not always *replace* the population, but *augment* it.

EDA pseudocode

Taken from [12]

```
EDA($f$, $S$)
$t \leftarrow 0$
generate initial population $P_0$
do
    evaluate solutions in $P_0$ using $f$
    select a sample of good solutions $S_t$ from $P_t$
    build a probabilistic model $M_t$ for $S_t$
    sample $M_t$ to generate new candidate solutions $O_t$
    incorporate $O_t$ into $P_t$
    $t \leftarrow t+1$
while(StoppingCondition($P_t$))
```

Note:

- In fact, this template is not generic enough to implement all variants of EDA (for instance M_t is only transitory here).
- PD is intended to model only the good solutions.
- It is assumed that the search bias is now embodied by the drawing and updating method.

The zoo of EDAs

The representation of PD determines (to some extent) the way it can be updated and used for generation of new solutions.

Variants of EDA:

- For **independent** variables
  - Univariate Marginal Distribution Algorithm (UMDA)
  - Population Based Incremental Learning (PBIL)
  - Compact Genetic Algorithm (CGA, Goldberg)
- For **bivariate** (pairwise) dependencies
  - Mutual Information-Maximizing Input Clustering (MIMIC, De Bonet et. al. 1997)
- For **multivariate** (arbitrary) dependencies
  - Bayesian Optimization Algorithm (BOA, Pelikan)
  - Hierarchical Bayesian Optimization Algorithm (HBOA, Pelikan)

EDAs for independent variables

Univariate Marginal Distribution Algorithm (UMDA)

The simplest form of EDA:

- For n-bit binary strings (n binary variables):
  - PD model: vector of probabilities $\mathbf{p} = (p_1, p_2, \ldots, p_n)$, where p_i is the probability of '1' for ith variable.
- Sampling the model: generate 1 for ith variable with probability p_i (independently).
- Learning the model $\Longrightarrow$

(Of course, trivially extensible to multi-valued variables).

UMDA: Learning the model

Compute frequency of 1s for particular positions

$$p_i = \frac{|\{\mathbf{x} \in S_t : x_i = 1\}|}{|S_t|}$$

Problem:

- For finite-sized population it may happen that $\{\mathbf{x} \in S_t : x_i = 1\} = \emptyset$, which implies $p_i = 0$ and no chance of ever having 1 on ith position.
- Solution: Laplace correction ('smoothing' of the estimate):

$$p_i = \frac{|\{\mathbf{x} \in S_t : x_i = 1\}| + 1}{|S_t| + |D(x_i)|}$$

where $D(x_i)$ is the domain of variable x_i (here: $D(x_i) = \{0, 1\}$, so $D(x_i) = 2$).

UMDA on OneMax

A classical, easy-to-solve optimization benchmark.

- The objective: generate a solution composed of all ones:

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

- A single global optimum: 1111...1
- Fully decomposable problem: how a given variable x_i contribute to solution's fitness is purely independent of what happens to the other variables.

UMDA: Very good performance:

- Expected convergence in $O(n \log n)$ steps
- Ideal scaling

PBIL: Taking history into account

Observation:

- UMDA builds PD from scratch in every generation.
- The knowledge about the previous generations is present only implicitly.

Why not accumulate that knowledge over multiple generations?

Population Based Incremental Learning (PBIL)

$$p_i^{(t)} = (1-\alpha)p_i^{(t-1)} + \alpha \frac{|\{\mathbf{x} \in S_t : x_i = 1\}|}{|S_t|}$$

where $\alpha \in (0,1)$.

UMDA on a deceptive problem

Trap-5 problem:

- Variables partitioned into disjoint subsets of size 5.

x_1				x_5	x_6				x_{10}	x_{11}				x_{15}	$\dots$

- For each group $\mathbf{x}'$, its contribution to fitness is defined as:

$$f_{contr}(\mathbf{x}') = \begin{cases} 5 & \textit{if } \mathbf{1}(\mathbf{x}') = 5 \\ 4 - \mathbf{1}(\mathbf{x}') & \textit{otherwise} \end{cases}$$

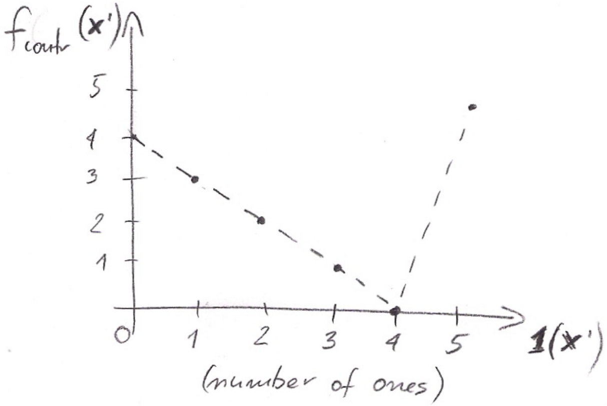
where $\mathbf{1}(\mathbf{x}')$ is the number of ones in $\mathbf{x}'$.

Properties:

- The global optimum is still 1111...1, but there is a negative slope towards it.
- Non-decomposable problems: contribution of a single variable depends on how the other variables in a group behave.

Trap-5

Fitness contribution $f_{contr}(\mathbf{x}')$ as a function of the number of ones $\mathbf{1}(\mathbf{x}')$



Trap-5

$\mathbf{x}'$					$f_{contr}(\mathbf{x}')$
1	1	1	1	1	5
0	1	1	1	1	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	0
0	0	1	1	1	1
			…		
0	0	0	1	1	2
			…		

- $\mathbb{E}(f_{contr}(0****)) = 2$
- $\mathbb{E}(f_{contr}(1****)) = 1.375$
- The algorithm is 'tempted' to generate zeroes.

UMDA on Trap-5

- Poor performance.
- Marginal distributions, each associated with a single variable, cannot capture the interdependencies between variables.

Remedies?

- Use PD model that is not marginal and thus can capture the interdependencies.
- E.g., for Trap-5: store probabilities for each combination of bits: $p(00000)$, $p(00001)$, $p(00010)$, ...
- Leads to very good performance again.
- However: intractable for large number and/or multivalued of variables.
- General conclusion: more sophisticated PD models needed to model the ‘context’ for each variable.

Skeptic's comment

Wait a minute: good performance could be probably equally well attained by designing an appropriate crossover operator, which would keep the groups of variables together.

Answer:

- Indeed, *designing*.
  - This stance assumes that the researcher *knows in advance* what is the optimal grouping of variables (what are the interactions between them).
  - For Trap-5, this is true.
  - For real-world problem, it is rarely the case.
- EDAs have the chance of autonomously discovering such groups (building blocks).
  - In a sense, and EDA *learns* a search operator while doing the search.

This the **key rationale** for EDAs.

EDAs for pairwise dependencies

EDAs for pairwise dependencies

Model **pairwise** dependencies.

- COMIT (Baluja & Davies 1997):
  - Uses a tree to model to capture the distribution
  - The tree is built so as to maximize the mutual information (minimize Kullback-Leiblerdivergence)
  - Algorithm: Prim's algorithm for maximum spanning trees.
- Similar:
  - BMDA (Pelikan, Mühlenbein, 1998)
  - MIMIC (DeBonet, 1996)

We skip these due to limited time.

EDAs for arbitrary dependencies

EDAs for arbitrary dependencies

Problem: Modeling a multivariate distribution for a large number of dimensions (variables) is challenging.

- Curse of dimensionality: for n variables, one needs $O(\prod_{i=1}^{n} D(x_i))$ examples (individuals) for the sample to be representative.

The most appealing solution for today: Bayesian Optimization Algorithm (BOA)

- Pelikan, Goldberg, & Cantú-Paz (1998)
- The idea: employ Bayesian Network (BN) to model the PD.

Bayesian Network

A graphical model of conditional dependencies (or: ‘conditional independence assertions’), which implicitly defines full joint distribution.

Two components:

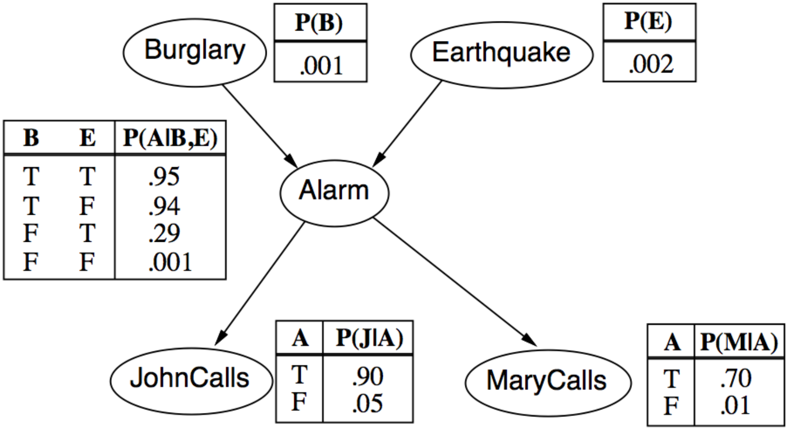
- Structure: directed acyclic graph
  - Nodes correspond to variables.
  - Edges correspond to **direct** conditional dependencies.
- Parameters
  - Set of conditional probability tables (CPTs), one per node. Specifies conditional probability that a variable takes on a value **given** the value of node’s predecessors.

$$p(X) = \prod_{i=1}^{n} p(X_i|\Pi_i)$$

where:

- $X = (X_1, \ldots, X_n)$ is the vector of all variables of the problem.
- Π_i is the set of parents of node X_i in the network

Exemplary BN



P(B)
.001

P(E)
.002

B	E	P(A|B,E)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J|A)
T	.90
F	.05

A	P(M|A)
T	.70
F	.01

(Taken from AIMA by Russel & Norvig).

What do we gain using BNs?

BNs enable specifying full PD using multiple conditional PDs.

- Usually, each variable depends only on **some** of the other variables.
- Assume each variable has at most k parents.
- The complete network requires then $O(n2^k)$ parameters. The number of parameters grows **linearly** with n.
- ... whereas the full joint distribution requires $O(2^n)$ parameters (a single number for each combination of variables).

Learning BNs

Two stages:

1. Learning the structure
2. Learning the parameters

In EDAs (in BOA), once the structure is learned, learning the parameters is easy, because there is no missing data: for each variable we have at least one ‘good’ value (remember that the model is built from good solutions only).

- Given a structure, learning of parameters consists in counting the relative frequencies for particular variables.

Learning BN structure

Learning BN from data (from sample) is difficult in general: finding the optimal structure is NP-complete for most ‘network quality metrics’.

Requires two elements:

- **Search algorithm**: searches the space of all possible network structures.
  - Greedy algorithms perform quite well.
  - Start from an empty network, and execute moves until no further improvement is possible:
    - Edge addition
    - Edge removal
    - Edge reversal
  - Warning: cycles have to be avoided!
- **Scoring metric**: should depend on the likelihood of the structure (plus sometimes a penalty term for too complex models). Examples:
  - Bayesian-Dirichlet metric
  - Minimum Description Length (MDL)

Are we losing something?

The BN induced from the population is **not optimal**, so it does not model perfectly the interdependencies between variables.

However:

- The conditional probabilities are calculated from a limited sample anyway, so they are not exact, only estimated (we do not have access to accurate conditional probabilities).
- The knowledge acquired in BN is subsequently exploited by a stochastic algorithm, which is not guaranteed to operate optimally.

So it's OK :) We are (probably) not loosing much.

Sampling BNs

The simplest method: probabilistic logic sampling.

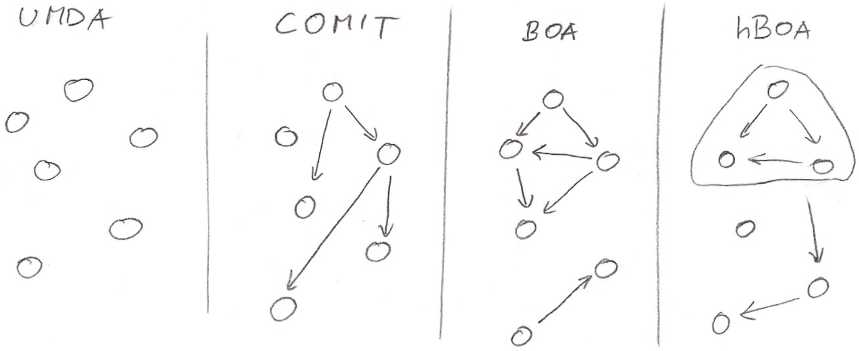
1. Compute the ancestral ordering of nodes (each node is preceded by its parents).
2. Generate the variable values according to the ancestral ordering (for every subsequent node, its parents already have the values assigned).

Note: It is sufficient to execute Step 1 only once for multiple solutions to be generated.

BOA's performance

- Scales very well
- A variant: Hierarchical BOA
  - Builds a hierarchical, multi-level BN.
  - Can effectively solve problems in which variables are hierarchically grouped (claimed to be frequent in real world).

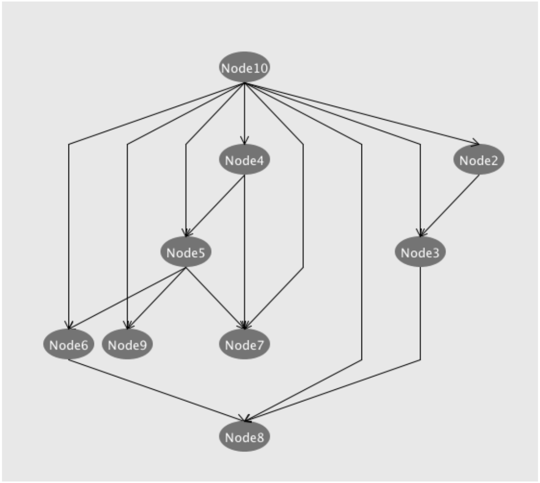
Summary of methods



Why are EDAs good (or should be good)?

- Formalization (population sizing,
- ‘Competence’ (Competent GAs, Competent EDAs).
- Scalability (?) and possibility to discover regularities in problems.
  - ‘Datamining’ of internal problem structure.
- Possibility of incorporating various methods from machine learning (ML).
  - E.g., ready-to-use implementations of BNs in WEKA and R.
- Combines learning (directed search) with evolution (fitness-based, less direct stochastic search)

Building BNs from data in WEKA



(WEKA enables also generation of data from randomly built BNs)

Applications

- Finding ground states of Ising spin glases (cf. certain neural net models, e.g., Hopfield networks),
- Maximum satisfiability (MAXSAT): find the assignment of values to Boolean variables such that satisfies the maximum number of conjuncts (clauses) in the conjunctive normal form (CNF)
    - attains performance comparable to WalkSat (in terms of the number of evaluations) on instances of hundreds of variables and thousands of conjuncts.

Things not considered in this course

- Theoretical developments (nice convergence proofs and recommendation for population sizing)
- EDAs for continuous problems (must discretize, or use PDFs)
- EDAs for other domains (not variable-based), e.g.: permutations or programs (GP)
- Multiobjective EDAs

Resources

Main source of information: `http://medal-lab.org/`

- Some papers and software available from there.
- For BNs, lots of good tutorials available online (including AIMA).

Recommended reading

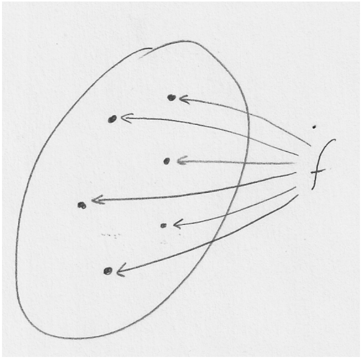
1. M. Pelikan. *Hierarchical Bayesian Optimization Algorithms*. Springer Verlag, 2005
2. M. Pelikan, K. Sastry, and E. Cantú-Paz, editors. *Scalable Optimization via Probabilistic Modeling*, volume 33 of *Studies in Computational Intelligence*. Springer, 2006
3. S. Luke. *Essentials of Metaheuristics*. lulu.com, first edition, 2009. Available at http://cs.gmu.edu/∼sean/books/metaheuristics/

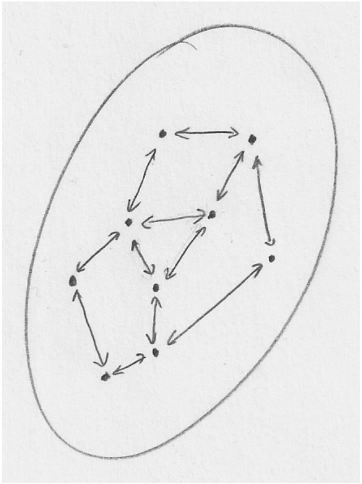
Part II: Coevolutionary algorithms

Coevolutionary algorithms

- A form of evolutionary algorithm where evaluation of individuals is influenced (determined) by other evolving individuals [5, p. 2].
  - (while in standard evolutionary algorithms, individuals interact only with the environment embodied by the fitness function f)
- The particular form of influence depends on the variant of coevolution.
- Essential feature of coevolution: **inter-individual interactions**.

Evolution vs. coevolution





Consequences

- The outcome of evaluation depends on who is the other participant of interaction.
- The individuals one interacts with form a **context**.
  - That context changes with time (from generation to generation).
  - Moreover, it can ‘respond’ to individual’s changes, because:
    - The outcome of my interaction with individual x influences it fitness.
    - Depending on that fitness, x can spawn an offspring in the next generation or not.
    - This in turn influences chances of survival for my offspring.
- Performing well or badly in a specific *context* does not mean being *objectively* good or bad.

Inspiration

- Nature knows only coevolution!
    - There is no ‘overlord’ that assigns fitness to each individual.
    - In biology, fitness is a quantity that can be measured, but not imposed.
    - E.g.: Absolute fitness of a genotype is the ratio between the number of individuals with that genotype after selection to those before selection [Wikipedia]
- The natural coevolution takes place at different levels:
    - intra-species, resulting from competition between individuals,
    - inter-species, resulting from competition between species, demes, etc.

Objective vs. subjective fitness

To implement section pressure, we still need some fitness function.

- Evolutionary algorithm uses *objective fitness* $f : \mathbb{S} \rightarrow \mathbb{R}$
- Coevolutionary algorithms (typically) use *subjective fitness* f_s, which is derived from the outcomes of interactions between individuals.
- Exemplary definition of subjective fitness:

$$f_s(s) = \sum_{s' \in P} g(s, s')$$

where $g : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}$ - an *interaction function*; more precisely:

$$f_s^{(t)}(s) = \sum_{s' \in P^{(\mathbf{t})}} g(s, s')$$

Notes:

- Typically, $f_s \neq f$
- The codomain of f and f_s does not have to be real-valued, any totally ordered set would do.

Isn't this odd?

Variants of coevolutionary algorithms

Various genres of coevolutionary algorithms differ mostly in the way the individuals interact with each other.

- Single-population vs. multi-population
  - Single-population: individuals kept in a single set, and interact with each other according to some scheme (e.g., round robing, k random opponents, single elimination tournament). All individuals play the same ‘roles’.
  - Multi-population: individuals split into two or more subsets. Interactions typically take place only between individuals from different populations. Individuals from different population may play different roles.
- Competitive vs. cooperative
  - Competitive: Individuals try to ‘win’ as many interactions as possible.
  - Cooperative: Individuals are assumed to cooperate to solve the posed task (typically used with multiple populations).

Single-population competitive coevolution

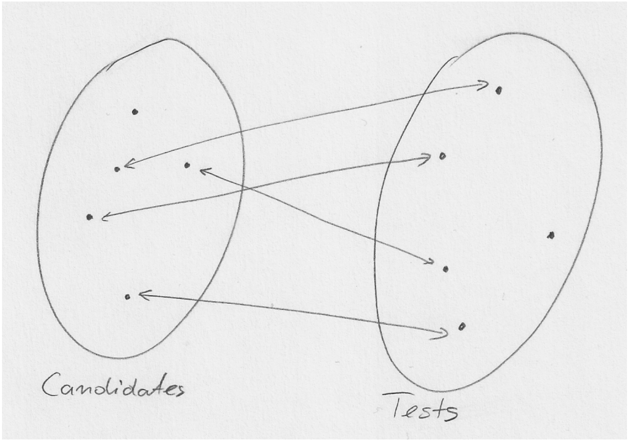
- Individuals compete against each other.
- In the evaluation phase, *tournaments* (contests) are being organized, in which pairs or groups of individuals from populations compete in interactions.
- The interaction function g typically implements a zero-sum game: the winner scores x while the loser $-x$
- A class of such problems is sometimes referred to as *adversarial problems* [16, p. 2].

Case study 1: Competitive coevolution

- Typical application of single-population competitive coevolution: learning game strategies.
    - Each individual implements a strategy.
    - Interaction consists in playing a game.
    - The result of the game (qualitative or quantitative) becomes the outcome of interaction.
    - Individual's fitness is the average outcome of all games played.
    - Successfully applied to games like checkers, Othello, ...
- Example (Case study 1):
    - Single-population fitnessless coevolution.
    - Key idea:
        - Standard approach: 1) play games, 2) calculate fitness, 3) use that fitness for selection.
        - Idea: combine 1) with 3), skipping 2). There is no explicit evaluation phase. Individuals play games and this determines whether they get selected or not.
    - Based on: W. Jaśkowski, K. Krawiec, and B. Wieloch. Evolving strategy for a probabilistic game of imperfect information using genetic programming. *Genetic Programming and Evolvable Machines*, 9(4):281–294, 2008

Two-population competitive coevolution

- Individuals partitioned into disjoint collections, called *subpopulations* P_i (*species* or *demes* in evolutionary theory). For brevity, we call them *populations*.



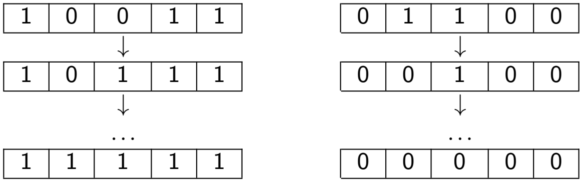
Two-population competitive coevolution

- One of motivations: asymmetry of interactions (interaction participants play different **roles**).
  - Example: white players and black players in checkers.
- Populations called also [1]:
  - ‘predators’ and ‘preys’,
  - ‘parasites’ and ‘hosts’,
  - ‘problem generators’ and ‘problem solvers’,
  - ‘teachers’ and ‘learners’,
  - ‘candidates’ and ‘tests’ .

An example: Density classification task (DCT)

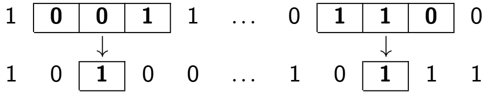
Density classification task: synthesize a **state transition rule** for a binary, one-dimensional cellular automaton (CA) of size *n*, such that it 'classifies' the initial state of the automaton depending on whether there are more 0's or 1's in it.

- If the initial state contains more 0's, the rule should transform the state into 'all 0's'.
- And vice versa for 1's.



Rules for automata

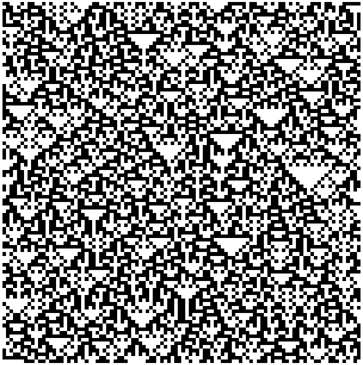
- Rules have fixed, limited radius r
  - $2r+1 \ll n$: rules cannot 'see' the entire state
- E.g., for $r=1$:



- Rules are represented as lookup tables
  - 2^{2r+1} entries.

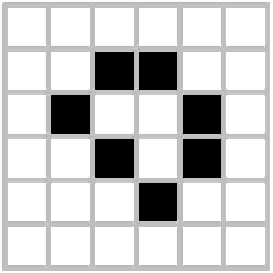
Digression 1: 1D automata

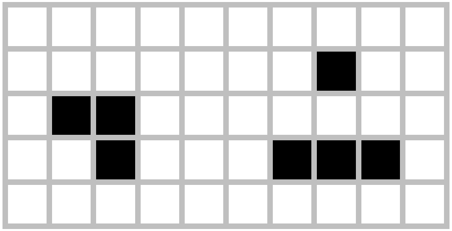
- Even simple rules can produce patterns of exquisite complexity.
  - Some of these rules are proven to perform Turing-complete computation.
- Wolphram's book "A New Kind of Science"



Digression 2: 2D automata

John Convay's Game of Life
http://en.wikipedia.org/wiki/Conway's_Game_of_Life





Complexity of DCT

- Exact evaluation of a rule requires applying it to all 2^n possible states.
- The most popular setting is $n = 149$, and the upper limit of the number of rule application 320.
- Posed as test-based problems:
  - Candidate solutions = transition rules
  - Tests = initial states of the automaton.
- The best rule for $n = 149$, $t = 320$, $r = 3$, found using coevolutionary algorithm: success rate 86.3% [6]; see also [2]
- **Key insight:** coevolutionary algorithm can reduce the number of interactions required to find a solution.
- See

Coevolutionary pathologies

Does it always work so well?

No. Lack of external, objective evaluation makes it difficult to predict where the evolution will head to.

Coevolutionary pathology – a situation in which solutions change with time (seem to ‘do’ something), but with no or little objective progress.

Selected types of pathologies (explained in next slides):

- Red Queen effect
- Disengagement
- Focusing (overfocusing)

Other: Collusion, Forgetting

Red Queen effect

Assume the is a cycle in the game graph. Example: Rock, Paper, Scissors game:

$$R \succ S,\ S \succ P,\ P \succ R$$

Consider the following scenario of evolution:

Generation (tour)	1	2	3	4	5	6	7	
Individual 1	R	R	P	P	S	S	R	…
Individual 2	S	R	R	P	P	S	S	…

- Both individuals do their best to beat the opponent, but overall there is no long-term progress.
- Lewis Carroll, *Through the Looking-Glass*:
    - *It takes all the running you can do, to keep in the same place.*
- Relative improvements (i.e., with respect to each other) do not translate into absolute (objective) improvement of fitness.
- Also known as *cycling*.
- Examples from nature: The arms race of parasites and host, immune system.

Cycling

- Example: Tic-tac-toe (TTT)
- Consider TTT strategies represented in a straightforward way: as an ordering of locations to be taken by a player.
- If a numbered location is already taken, the player places a piece on the location marked by '*'.

A

1	2	3
		*

B

3		
2		
1		*

C

		1
	2	
3		*

$$A \succ B,\ B \succ C,\ C \succ A$$

Note:

- These are simple examples, in real-world this becomes more subtle (and 'noisy').
- Certain variants of coevolution (e.g., fitnessless coevolution) cease to behave like evolution for intransitive interactions.

Disengagement

Applies mostly (if not exclusively) to two-population coevolution.

Example: Checkers

- Population P_1: White players
- Population P_2: Black players
- Assume P_1 is filled with master-level players, while P_2 contains only novices.
    - All interactions between players from P_1 and P_2 end with the former ones winning.
    - All masters seem to be equally good; all novices seem to be equally bad (even if they are in fact different!).
- There is no way to tell apart better masters from good masters; similarly for novices. Evolution stalls.
- A.k.a. *loss of gradient*.
- The above may happen during evolution, or may apply to initial state of evolution (how should I draw individuals for my initial population?)

Focusing

Checkers example cont'd.

- Assume individuals in P_1 converge so that they all start the game with moving the leftmost piece on the board (but possibly later play differently).
- The opponents from P_2 will 'get used' to it. Never having an opportunity to face an opponent that behaves differently, they will **specialize** in beating white players that start with the leftmost piece.
- When faced with new ('external') players (e.g., human), players from P_2 will be likely to lose.
- A.k.a. over-specialization.
- May be seen as an analog to overfitting in machine learning.

Lessons learned from coevolutionary pathologies

Problems arise when:

- Population(s) converge (focusing).
- Population(s) diverge (disengagement).
- Population(s) forget that they've 'already been there'.

The diagnosis:

- Pure coevolution has rather **bad memory**.
- Why does it work in Nature? Nature does not care about objective progress.

The problem

The current population is expected to:

- be diversified enough to enable further search and progress,
- represent (store?) the best solution found so far,

It can be difficult to do both at the same time.

The idea: split these functionalities, delegating (2) to an **archive**.

Archives

An archive is a 'memory' of a coevolutionary search.

- Maintains 'good' solutions found so far in the search process.
- Can be used to confront the evolved solutions with.
- [Sometimes] represents the final outcome of the search process.

Types of archives:

- Hall-of-fame
- Dominance tournament
- Nash memory
- Pareto archives

Hall-of-fame (HoF)

- Initially an empty set
- Maintenance: Extended by the (subjectively) best-of-generation in each generation
  - (grows indefinitely)
- Exploitation: Draw k members from HoF and let the individuals in the population interact (play) with them
- As a result, every individual plays with its peers and with some ‘older masters’.
- The outcomes of interactions with the HoF members [partially] influence fitness.

Rationale: A good individual should perform well against its peers in population *as well as the HoF members*.

- This provides a form of *historic progress* [9].

Pareto archives (Pareto coevolution)

First note some downsides of HoF:

- HoF is ‘passive’, never changes on its own.
- May contain many weak individuals (from the initial generations of the run), which may be not worth to interact with.

Note: The purpose of individuals in archive is not to perform good, but to **tell apart good and bad candidate solutions** (provide gradient for them).

The idea: Let the members of archive evolve too, but using **a different objective**.

- This takes [again] to [a variant of] two-population coevolution:
  - Population of candidate solutions (candidates),
  - Population of tests (archive)
- Example: Say we want evolve a white player’s strategy for checkers.
  - Candidates: white players.
  - Tests: black players.
  - Candidates get rewards for **performing** against tests, e.g., the number of wins against tests.
  - Tests get awards for **distinctions**, e.g., how many pairs of candidates they differentiate.

Test-based problems

Problems in which:

- Interaction function can be defined.
- Exact evaluation of solutions involves many (possibly infinitely many) interactions.

Domain	Candidate	Test
Algorithm design	Sorting network	Unsorted list
Classification	Classifier	Data point (or subset thereof)
Function regression	Function	Input (datapoint)
Strategy learning	First player	Second player
Optimization	Search algorithm	Problem instance

Solution concepts: What do we optimize?

- Given that the interaction function g is the only driving force of the search process, where does this process head to?
  - Can we identify somehow the goal of the search process?
  - What is the **solution**?
- The answer: **solution concept**: a subset of the search space that contains the solutions to be sought.
- Among many solution concepts, some are more useful/natural (see next slide).
- Various coevolutionary algorithms are designed with specific solution concepts in mind.

Selected solution concepts

Simultaneous maximization of all outcomes

- A solution belongs to this solution concept if it maximally beats all other solutions:

$$\{s \in S : \forall t \in S, t \neq s : g(s,t) = g_{max}\}$$

- Quite naive. Will be often empty.

Maximization of expected utility

- A solution belongs to this solution concept if it offers a maximal outcome of interaction against a randomly drawn opponent (context):

$$\arg\max_{s \in S} \mathbb{E}(g(s,t))$$

- where t is randomly drawn from S ($\arg\max$ may return a set).

Other well-defined solution concepts:

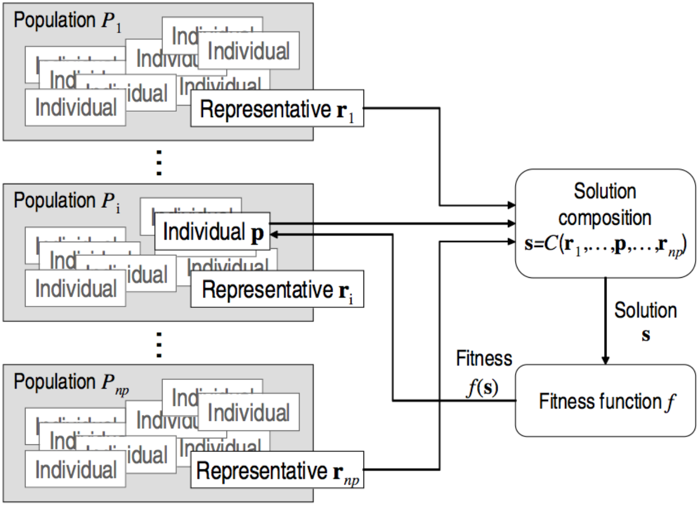
- Pareto-optimal set
- Best worse case

(See textbooks on game theory for more on that)

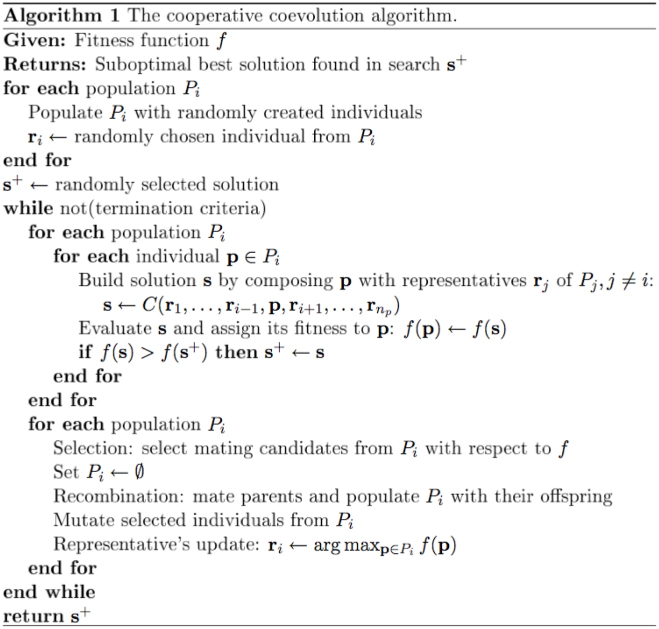
Cooperative coevolution

- Individuals from particular populations encode disjoint **parts** of the solution.
  - Requires a modular representation of the problem
  - Offers some means to decompose a complex problem.
- Typically, populations P_i, $i = 1 \ldots n_p$, are delegated to work on the i^{th} fragment of the whole solution.
- Referred also to as *symbiotic* [16, p.8] or *parasitic* [3] coevolution[3, 18, 14, 15]

Cooperative coevolution: The idea



Cooperative coevolution: The algorithm



Algorithm 1 The cooperative coevolution algorithm.

```
Given: Fitness function $f$
Returns: Suboptimal best solution found in search $\mathbf{s}^+$
for each population $P_i$
    Populate $P_i$ with randomly created individuals
    $\mathbf{r}_i \leftarrow$ randomly chosen individual from $P_i$
end for
$\mathbf{s}^+ \leftarrow$ randomly selected solution
while not(termination criteria)
    for each population $P_i$
        for each individual $\mathbf{p} \in P_i$
            Build solution $\mathbf{s}$ by composing $\mathbf{p}$ with representatives $\mathbf{r}_j$ of $P_j, j \neq i$:
                $\mathbf{s} \leftarrow C(\mathbf{r}_1, \ldots, \mathbf{r}_{i-1}, \mathbf{p}, \mathbf{r}_{i+1}, \ldots, \mathbf{r}_{n_p})$
            Evaluate $\mathbf{s}$ and assign its fitness to $\mathbf{p}$: $f(\mathbf{p}) \leftarrow f(\mathbf{s})$
            if $f(\mathbf{s}) > f(\mathbf{s}^+)$ then $\mathbf{s}^+ \leftarrow \mathbf{s}$
        end for
    end for
    for each population $P_i$
        Selection: select mating candidates from $P_i$ with respect to $f$
        Set $P_i \leftarrow \emptyset$
        Recombination: mate parents and populate $P_i$ with their offspring
        Mutate selected individuals from $P_i$
        Representative's update: $\mathbf{r}_i \leftarrow \arg\max_{\mathbf{p} \in P_i} f(\mathbf{p})$
    end for
end while
return $\mathbf{s}^+$
```

Assignment

Assignment

I. Read **one** of the papers from the following list, focusing on the following issues:

- What is the question addressed in the paper?
- What data or evidence was collected by the author(s) to address the question?
- What did the data or evidence show?

II. Prepare a report (in English (preferably) or Polish) containing:

1. Your first and last name
2. Authors and the title of the paper
3. A few sentences about the strong (most interesting, intriguing) elements of the proposed approach
4. A few sentences about the weak points
5. Your individual thoughts/observations concerning the paper.
6. How could this be employed to solve some problems in your research area.

Email the report (plain text, no attachments!) to krawiec@cs.put.poznan.pl with "[SD]" tag in the email subject **by June 15th**.

Paper #1: EDA

M. Pelikan. Analysis of estimation of distribution algorithms and genetic algorithms on NK landscapes. *CoRR*, abs/0801.3111, 2008

Abstract: This study analyzes performance of several genetic and evolutionary algorithms on randomly generated NK fitness landscapes with various values of n and k. A large number of NK problem instances are first generated for each n and k, and the global optimum of each instance is obtained using the branch-and-bound algorithm. Next, the hierarchical Bayesian optimization algorithm (hBOA), the univariate marginal distribution algorithm (UMDA), and the simple genetic algorithm (GA) with uniform and two-point crossover operators are applied to all generated instances. Performance of all algorithms is then analyzed and compared, and the results are discussed.

`citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.152.9741`

Paper #2: Competitive coevolution (two-population)

S. Ficici and J. Pollack. Pareto optimality in coevolutionary learning. volume 2159, pages 316–325. Springer-Verlag, London, UK, 2001

Abstract: We develop a novel coevolutionary algorithm based upon the concept of Pareto optimality. The Pareto criterion is core to conventional multi-objective optimization (MOO) algorithms. We can think of agents in a coevolutionary system as performing MOO, as well: An agent interacts with many other agents, each of which can be regarded as an objective for optimization. We adapt the Pareto concept to allow agents to follow gradient and create gradient for others to follow, such that co-evolutionary learning succeeds. We demonstrate our Pareto coevolution methodology with the majority function, a density classification task for cellular automata.

`http://dl.acm.org/citation.cfm?id=757431`

Paper #3: Cooperative coevolution

M. Potter and K. D. Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000

Abstract: To successfully apply evolutionary algorithms to the solution of increasingly complex problems, we must develop effective techniques for evolving solutions in the form of interacting coadapted subcomponents. One of the major difficulties comes in finding computational extensions to our current evolutionary paradigms that will enable such subcomponents to "emerge" rather than being hand designed. In this paper we describe an architecture for evolving such subcomponents as a collection of cooperating species. Given a simple string-matching task, we show that evolutionary pressure to increase the overall fitness of the ecosystem can provide the needed stimulus for the emergence of an appropriate number of interdependent subcomponents that cover multiple niches, evolve to an appropriate level of generality, and adapt as the number and roles of their fellow subcomponents change over time. We then explore these issues within the context of a more complicated domain through a case study involving the evolution of artificial neural networks.

`portal.acm.org/citation.cfm?id=1108890`

Paper #4: GP

M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 3 Apr. 2009

Abstract: For centuries, scientists have attempted to identify and document analytical laws that underlie physical phenomena in nature. Despite the prevalence of computing power, the process of finding natural laws and their corresponding equations has resisted automation. A key challenge to finding analytic relations automatically is defining algorithmically what makes a correlation in observed data important and insightful. We propose a principle for the identification of nontriviality. We demonstrated this approach by automatically searching motion-tracking data captured from various physical systems, ranging from simple harmonic oscillators to chaotic double-pendula. Without any prior knowledge about physics, kinematics, or geometry, the algorithm discovered Hamiltonians, Lagrangians, and other laws of geometric and momentum conservation. The discovery rate accelerated as laws found for simpler systems were used to bootstrap explanations for more complex systems, gradually uncovering the "alphabet" used to describe those systems.

`www.sciencemag.org/content/324/5923/81.short`

(plus accompanying material)

Paper #5: Applied GP

W. Weimer, S. Forrest, C. Le Goues, and T. Nguyen. Automatic program repair with evolutionary computation. *Communications of the ACM*, 53(5):109–116, June 2010

Abstract: There are many methods for detecting and mitigating software errors but few generic methods for automatically repairing errors once they are discovered. This paper highlights recent work combining program analysis methods with evolutionary computation to automatically repair bugs in off-the-shelf legacy C programs. The method takes as input the buggy C source code, a failed test case that demonstrates the bug, and a small number of other test cases that encode the required functionality of the program. The repair procedure does not rely on formal specifications, making it applicable to a wide range of extant software for which formal specifications rarely exist.

`dl.acm.org/ft_gateway.cfm?id=1735249&type=html`

Paper #6: Modularity

R. A. Watson and J. B. Pollack. Modular interdependency in complex dynamical systems. *Artif. Life*, 11(4):445–458, 2005

Abstract: Herbert A. Simon's characterization of modularity in dynamical systems describes subsystems as having dynamics that are approximately independent of those of other subsystems (in the short term). This fits with the general intuition that modules must, by definition, be approximately independent. In the evolution of complex systems, such modularity may enable subsystems to be modified and adapted independently of other subsystems, whereas in a nonmodular system, modifications to one part of the system may result in deleterious side effects elsewhere in the system. But this notion of modularity and its effect on evolvability is not well quantified and is rather simplistic. In particular, modularity need not imply that intermodule dependences are weak or unimportant. In dynamical systems this is acknowledged by Simon's suggestion that, in the long term, the dynamical behaviors of subsystems do interact with one another, albeit in an "aggregate" manner—but this kind of intermodule interaction is omitted in models of modularity for evolvability. In this brief discussion we seek to unify notions of modularity in dynamical systems with notions of how modularity affects evolvability. This leads to a quantifiable measure of modularity and a different understanding of its effect on evolvability.

`http://www.mitpressjournals.org/doi/pdf/10.1162/106454605774270589`

Paper #7: Modularity, EA

N. Kashtan and U. Alon. Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences*, 102(39):13773–13778, Sept. 27 2005

Abstract: Biological networks have an inherent simplicity: they are modular with a design that can be separated into units that perform almost independently. Furthermore, they show reuse of recurring patterns termed network motifs. Little is known about the evolutionary origin of these properties. Current models of biological evolution typically produce networks that are highly nonmodular and lack understandable motifs. Here, we suggest a possible explanation for the origin of modularity and network motifs in biology. We use standard evolutionary algorithms to evolve networks. A key feature in this study is evolution under an environment (evolutionary goal) that changes in a modular fashion. That is, we repeatedly switch between several goals, each made of a different combination of subgoals. We find that such modularly varying goals lead to the spontaneous evolution of modular network structure and network motifs. The resulting networks rapidly evolve to satisfy each of the different goals. Such switching between related goals may represent biological evolution in a changing environment that requires different combinations of a set of basic biological functions. The present study may shed light on the evolutionary forces that promote structural simplicity in biological networks and offers ways to improve the evolutionary design of engineered systems.

`www.pnas.org/content/102/39/13773.abstract`

Bibliography

A. Bucci and J. Pollack.
A mathematical framework for the study of coevolution.
In K. D. Jong, R. Poli, and J. Rowe, editors, *Foundations of Genetic Algorithms 7*, pages 221–235, San Francisco, 2003. Morgan Kaufmann.

S. Ficici and J. Pollack.
Pareto optimality in coevolutionary learning.
volume 2159, pages 316–325. Springer-Verlag, London, UK, 2001.

W. Hillis.
Co-evolving parasites improve simulated evolution as an optimization procedure.
In C. Langton et al., editors, *Artificial Life II*. Addison-Wesley, 1991.

W. Jaśkowski, K. Krawiec, and B. Wieloch.
Evolving strategy for a probabilistic game of imperfect information using genetic programming.
Genetic Programming and Evolvable Machines, 9(4):281–294, 2008.

E. D. Jong and T. Oates.
A coevolutionary approach to representation development.
In *Proc. International Conference on Machine Learning, Workshop on Development of Representations*, 2002.

H. Juille and J. B. Pollack.
Coevolving the ideal trainer: Application to the discovery of cellular automata rules.
In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 519–527, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

N. Kashtan and U. Alon.
Spontaneous evolution of modularity and network motifs.
Proceedings of the National Academy of Sciences, 102(39):13773–13778, Sept. 27 2005.

S. Luke.
Essentials of Metaheuristics.
lulu.com, first edition, 2009.
Available at http://cs.gmu.edu/∼sean/books/metaheuristics/.

T. Miconi.
Why coevolution doesn't "work": superiority and progress in coevolution.
In L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, and M. Ebner, editors, *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, volume 5481 of *LNCS*, pages 49–60, Tuebingen, Apr. 15-17 2009. Springer.

M. Pelikan.
Hierarchical Bayesian Optimization Algorithms.
Springer Verlag, 2005.

M. Pelikan.
Analysis of estimation of distribution algorithms and genetic algorithms on NK landscapes.
CoRR, abs/0801.3111, 2008.

M. Pelikan.
Probabilistic model-building genetic algorithms.
In C. Blum, editor, *GECCO 2011 Late breaking abstracts*, pages 913–940, Dublin, Ireland, 12-16 July 2011. ACM.

M. Pelikan, K. Sastry, and E. Cantú-Paz, editors.
Scalable Optimization via Probabilistic Modeling, volume 33 of *Studies in Computational Intelligence*.
Springer, 2006.

M. Potter.
The Design and Analysis of a Computational Model of Cooperative Coevolution.
PhD thesis, George Mason University, 1997.

M. Potter and K. D. Jong.
Cooperative coevolution: An architecture for evolving coadapted subcomponents.
Evolutionary Computation, 8(1):1–29, 2000.

C. Rosin.
Coevolutionary Search Among Adversaries.
PhD thesis, University of California, San Diego, 1997.

M. Schmidt and H. Lipson.
Distilling free-form natural laws from experimental data.
Science, 324(5923):81–85, 3 Apr. 2009.

R. Smith, S. Forrest, and A. Perelson.
Searching for diverse, cooperative populations with genetic algorithms.
Evolutionary Computation, 1(2), 1993.

R. A. Watson and J. B. Pollack.
Modular interdependency in complex dynamical systems.
Artif. Life, 11(4):445–458, 2005.

W. Weimer, S. Forrest, C. Le Goues, and T. Nguyen.
Automatic program repair with evolutionary computation.
Communications of the ACM, 53(5):109–116, June 2010.