Algorytmy ewolucyjne i ich zastosowania w sztucznej inteligencji

Krzysztof Krawiec
Instytut Informatyki, Politechnika Poznańska
Center of excellence in Artificial Intelligence
and Machine Learning

Seminarium interdyscyplinarne
"Paradygmat ewolucji w naukach społecznych"
Katedra Konkurencyjności Międzynarodowej UEP

26.11.2019

Agenda

1. Problemy przeszukiwania i problemy optymalizacyjne
2. Algorytmy rozwiązywania problemów optymalizacyjnych
3. Algorytmy ewolucyjne
4. Algorytmy koewolucyjne
5. Programowanie genetyczne
6. Uwagi końcowe

1.
Problemy przeszukiwania i problemy optymalizacyjne

Problemy przeszukiwania i problemy optymalizacyjne

Problem przeszukiwania:

> *Mając dany zbiór wszystkich możliwych (dopuszczalnych) rozwiązań X, znajdź rozwiązanie x spełniające pewną właściwość P(x).*

Uwagi:

- X jest zazwyczaj zadany *implicite* (przez określenie *dziedziny* i *ograniczeń*).
- Zbiór X może być, zależnie od typu problemu
  - Ciągły lub dyskretny
  - Skończony lub nieskończony
  - Rozpięty na produkcie Kartezjańskim lub nie.
- Właściwość P może być zadana jawnie, lub mieć charakter *wyroczni*.
- Rozwiązań spełniających P może być jedno, wiele, lub nieskończenie wiele.

Problemy przeszukiwania i problemy optymalizacyjne

Problem optymalizacyjne:

> *Mając dany zbiór wszystkich możliwych (dopuszczalnych) rozwiązań X, znajdź rozwiązanie x minimalizujące pewną funkcję celu f(x).*

Uwagi:

- Lub: maksymalizujące.
- Wartość f w optimum może być znana lub nieznana.
- Każdy problem optymalizacyjny może zostać przekształcony do problemu przeszukiwania poprzez zadanie pewnego *progu aspiracji* na f.

2.
Algorytmy rozwiązywania problemów optymalizacyjnych

Klasy algorytmów rozwiązujących problemy optymalizacyjne

- Algorytmy dokładne
  - Gwarantują znalezienie optimum.
  - Czas obliczeń może być wykładniczy względem rozmiaru instancji problemu.
- Algorytmy aproksymacyjne
  - Gwarantują znalezienie rozwiązania nie gorszego niż określony ‘procent’ jakości rozwiązania optymalnego.
  - Czas obliczeń wielomianowy.
  - Da się je zaprojektować jedynie dla pewnych klas problemów.
- Algorytmy heurystyczne
  - Nie dają żadnych gwarancji na znalezienie optimum ani na przybliżenie jego jakości.
  - Czas obliczeń wielomianowy.

3.
Algorytmy ewolucyjne

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

- 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)

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.

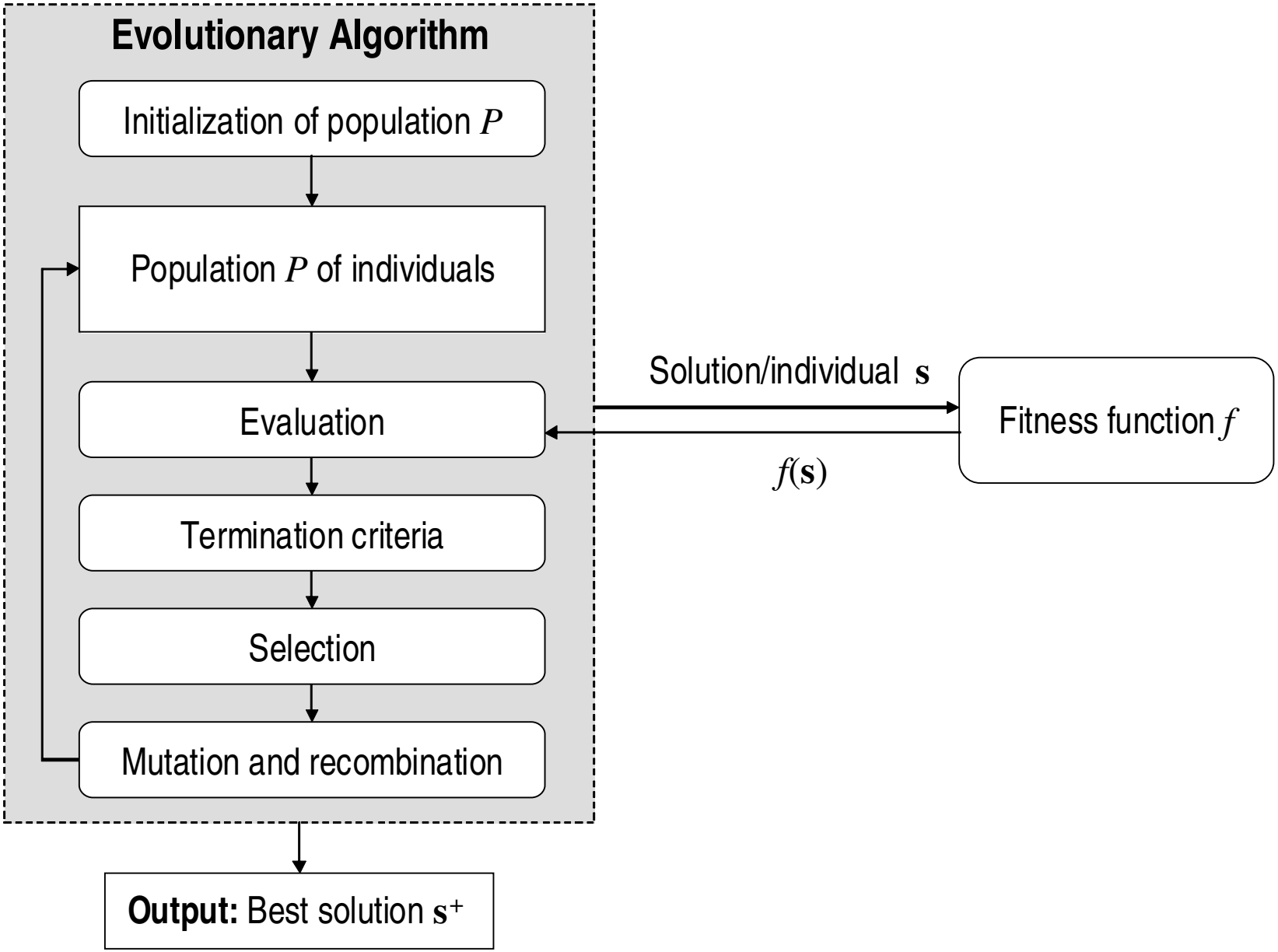
Formulation:

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

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



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?

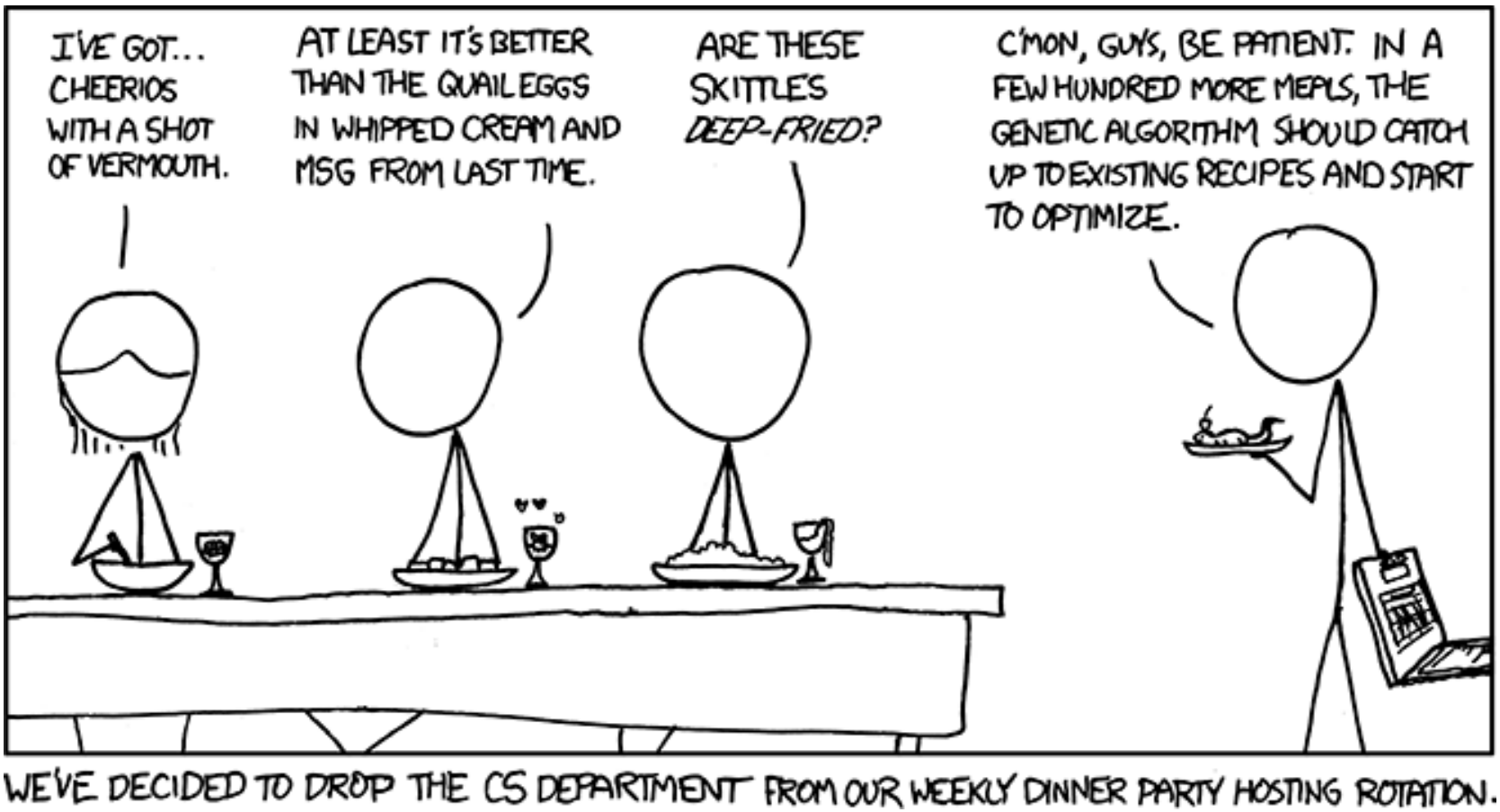
- Importance of **recombination** (crossover): a recombination operator that makes the solutions exchange certain elements (variable values, features)
- EC performs **global** search

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.

Convergence to good solutions may take some time ...



Source: http://xkcd.com/720/

(Actually, some variants of EC maintain and manipulate infeasible solutions)

Variants of evolutionary algorithms

Well rooted in EC:

- Genetic algorithms (GA): discrete (binary) encoding
- Evolutionary strategies (ES): real-valued encoding
- Evolutionary programming (EP): not particularly popular nowadays, but historically one of the first approaches to EC
- Genetic Programming (GP)

Newer branches:

- Estimation of distribution algorithms (EDA), generative and developmental systems (GDS), differential evolution, learning classifier systems, ...
- Not strictly EC: particle swarm optimization (PSO), ant colony optimization (ACO),

Note:

- EC = Evolutionary Computation, the name of the *domain*

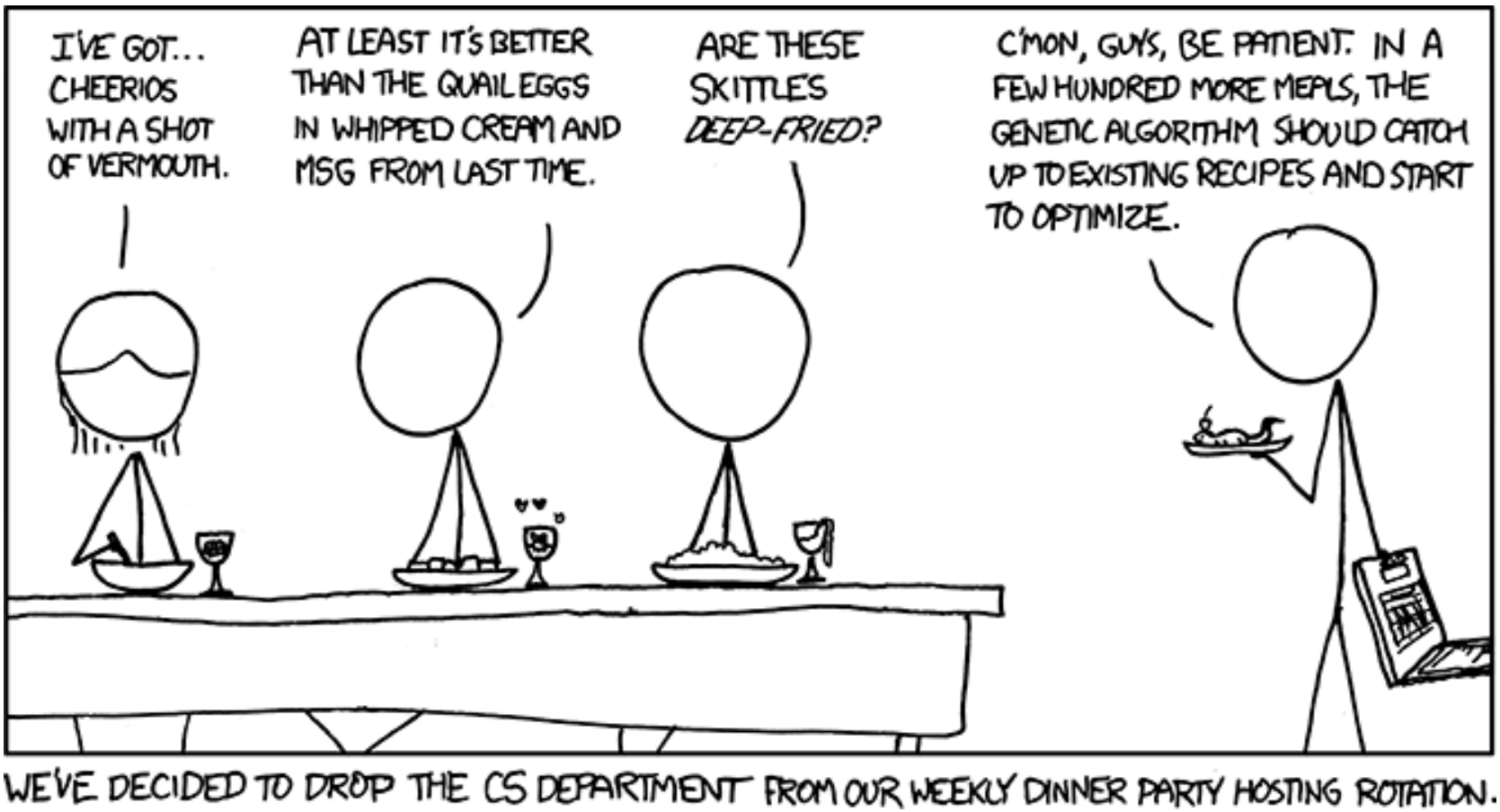
Major events of EC

- Genetic and Evolutionary Computation Conference (GECCO)
- IEEE Congress on Evolutionary Computation (CEC)
- EvoStar (Evo*)
- Parallel Problem Solving from Nature (PPSN)

Some facts:

- ACM SIGEVO group
- IEEE Task Forces
- Several dozens of thousands of publications (GP alone has almost 10,000)
- EC considered one of the three major branches of Computational Intelligence (Fuzzy Systems and Neural Nets being the other ones)

Convergence to good solutions may take some time ...



Source: http://xkcd.com/720/

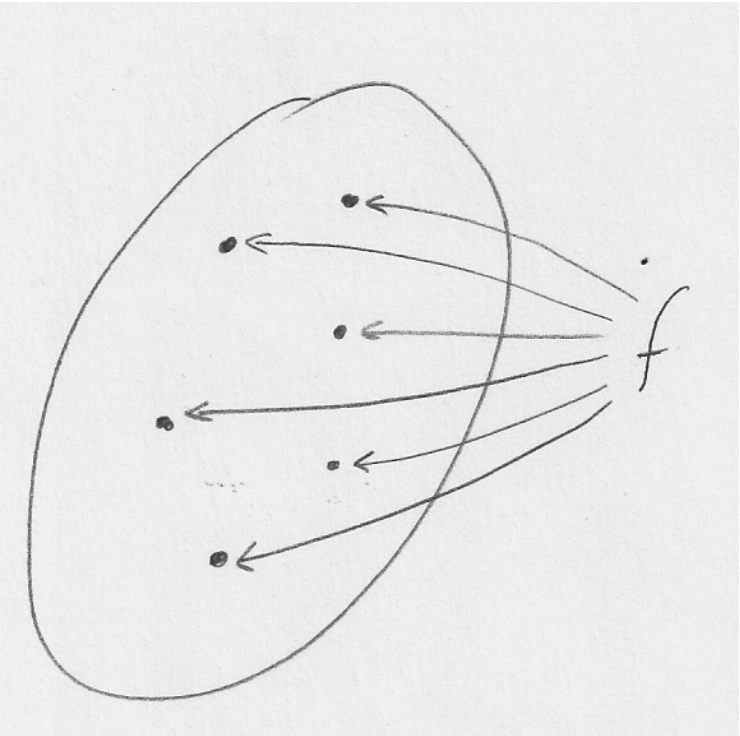
(Actually, some variants of EC maintain and manipulate infeasible solutions)

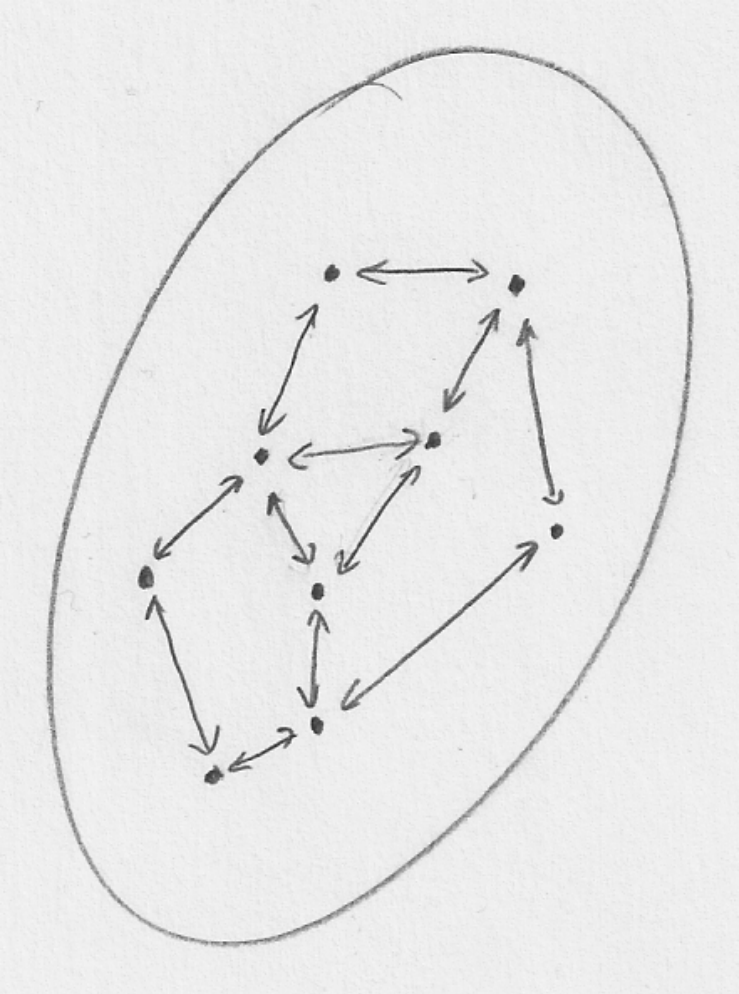
4.
Algorytmy koewolucyjne

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

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

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

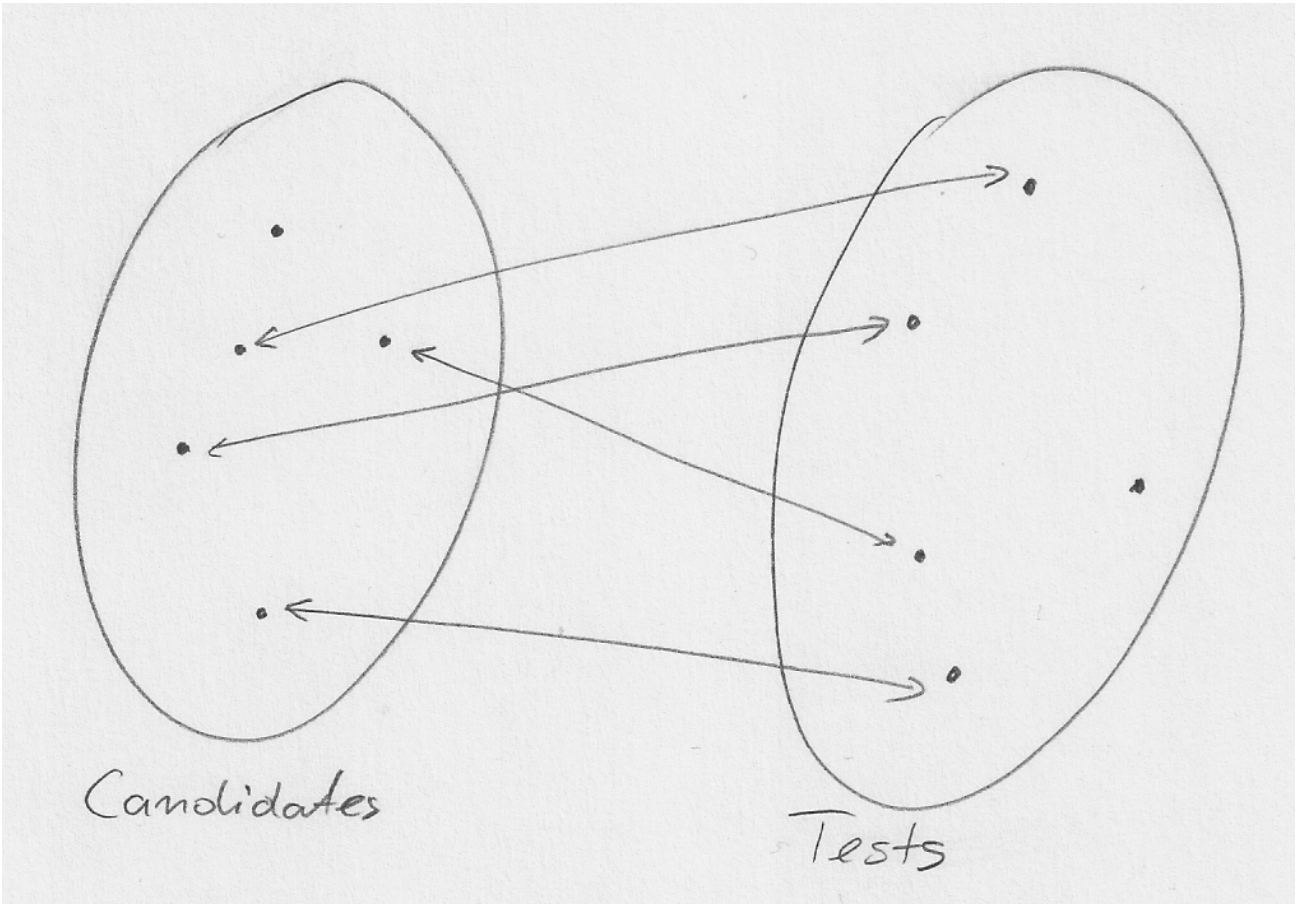
- Individuals compete against each other.
- In the evaluation phase, *tournaments* (contests) are being organized, in which pairs or groups of individuals from populations compete in interactions.
- The interaction function g typically implements a zero-sum game: the winner scores x while the loser $-x$
- 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*.



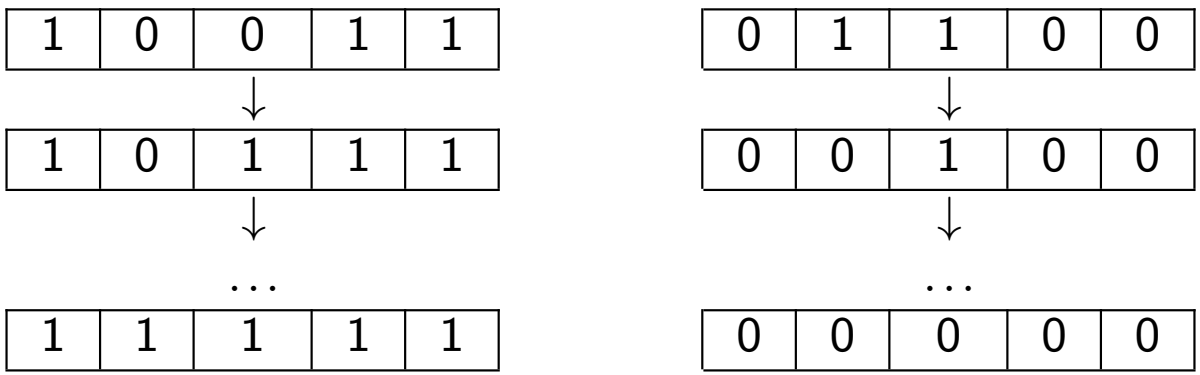
Two-population competitive coevolution

- 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)

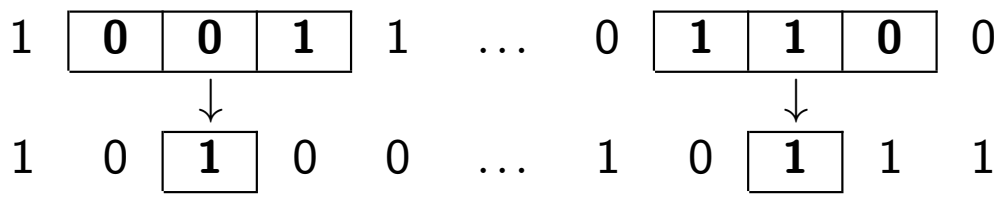
Density classification task: synthesize a **state transition rule** for a binary, one-dimensional cellular automaton (CA) of size n, such that it 'classifies' the initial state of the automaton depending on whether there are more 0's or 1's in it.

- 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

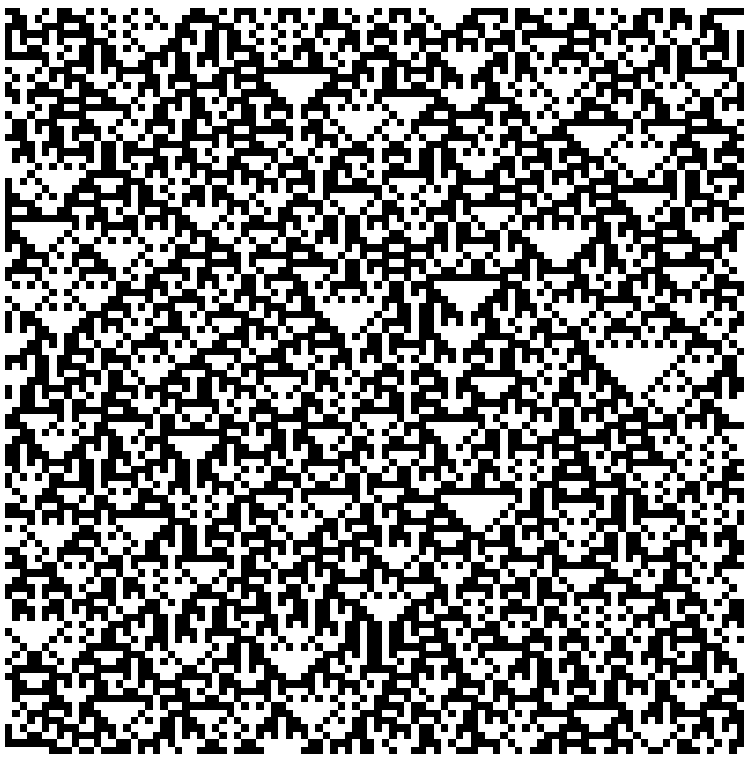
- Rules have fixed, limited radius r
  - $2r+1 \ll n$: rules cannot 'see' the entire state
- E.g., for $r = 1$:



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

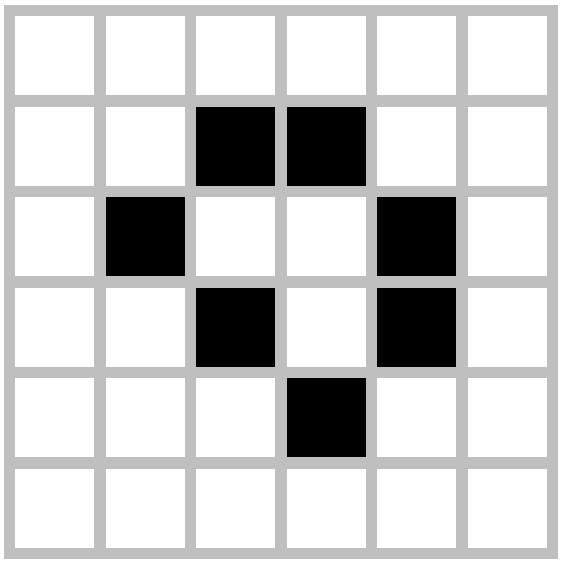
Digression 1: 1D automata

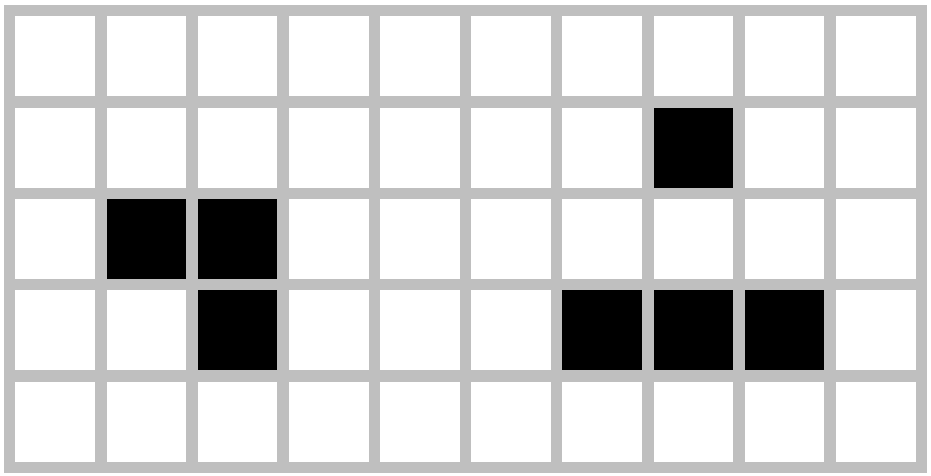
- Even simple rules can produce patterns of exquisite complexity.
  - Some of these rules are proven to perform Turing-complete computation.
- Wolphram's book "A New Kind of Science"



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:

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

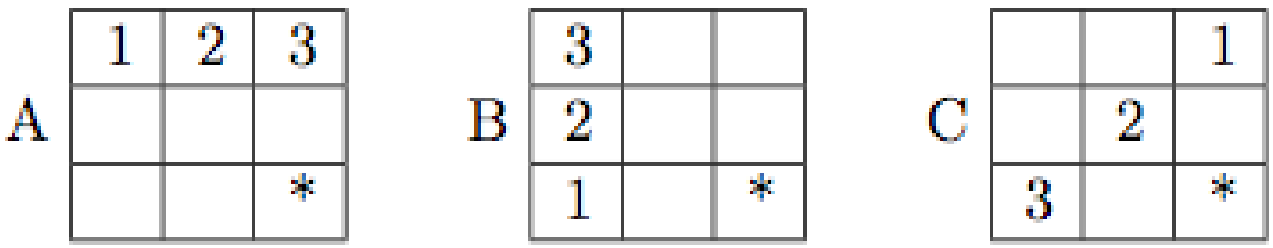
Consider the following scenario of evolution:

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 '*'.



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

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?)

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.

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:

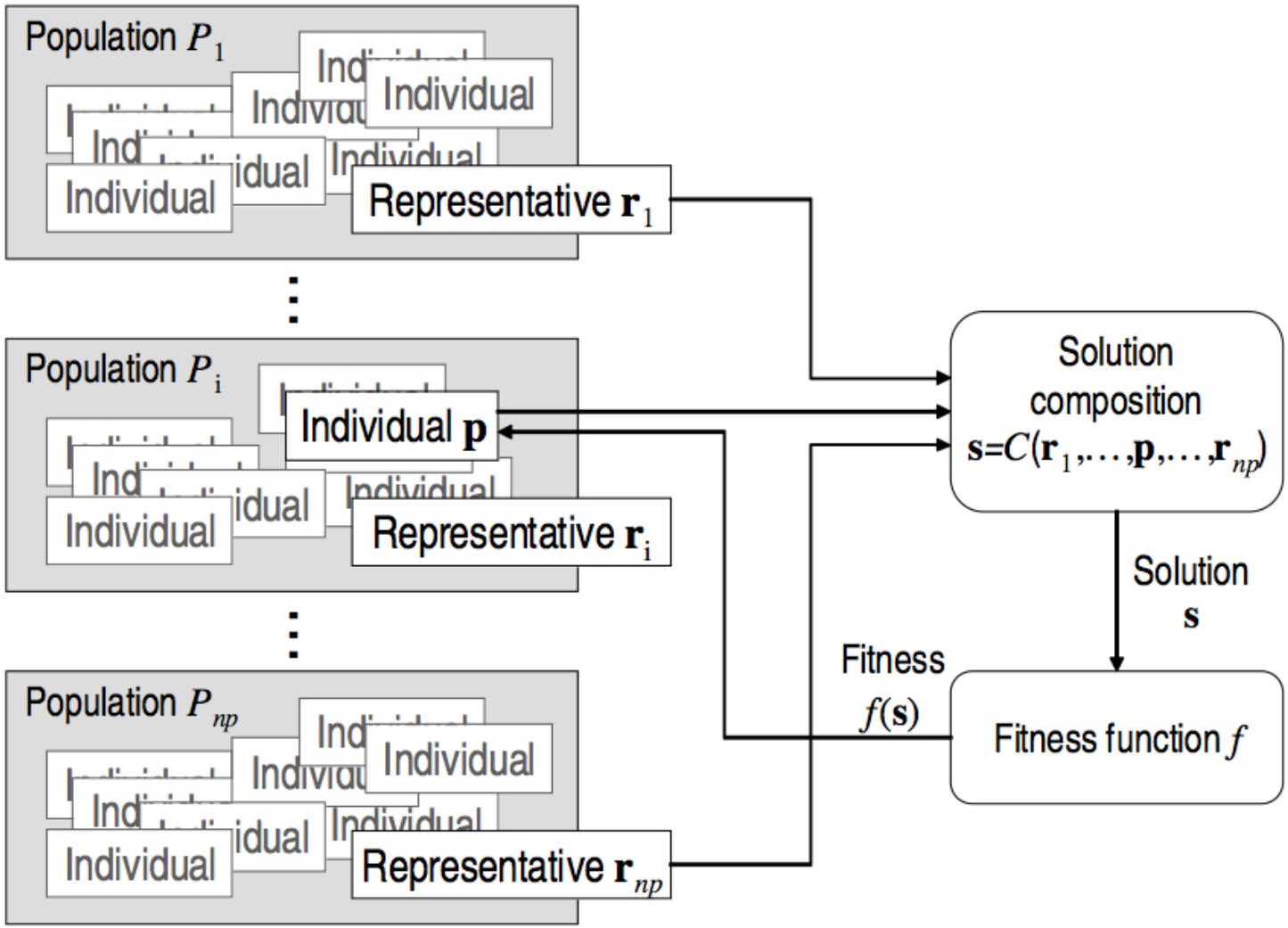
- Pareto-optimal set
- Best worse case

(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 $\mathbf{s}^+$

5.
Programowanie genetyczne

Genetic programming

In a nutshell:

- A variant of EA where the genotypes represent *programs*, i.e., entities capable of reading in input data and producing some output data in response to that input.
- The candidate solutions in GP are being assembled from elementary entities called *instructions*.
- Most common program representation: expression trees.
- Cardinality of search space large or infinite.

Fitness function

EA solves optimization problems. Program synthesis is a search problem. How to match them?

- Fitness function f measures the *similarity* of the output produced by the program to the desired output, given as a part of task statement.
- The set of program inputs I, even if finite, is usually so large that running each candidate solution on all possible inputs becomes intractable.
- GP algorithms typically evaluate solutions on a sample $I' \subset I, |I'| \ll |I|$ of possible inputs, and fitness is only an approximate estimate of solution quality.
- The task is given as a set of *fitness cases*, i.e., pairs $(x_i, y_i) \in I \times O$, where x_i usually comprises one or more independent variables and y_i is the output variable.

Fitness function: Example

City-block fitness function:

$$f(p) = -\sum_i ||y_i - p(x_i)||, \tag{1}$$

where

- $p(x_i)$ is the output produced by program p for the input data x_i,
- $||\cdot||$ is a metric (a norm) in the output space O,
- i iterates over all fitness cases.

Genetic programming

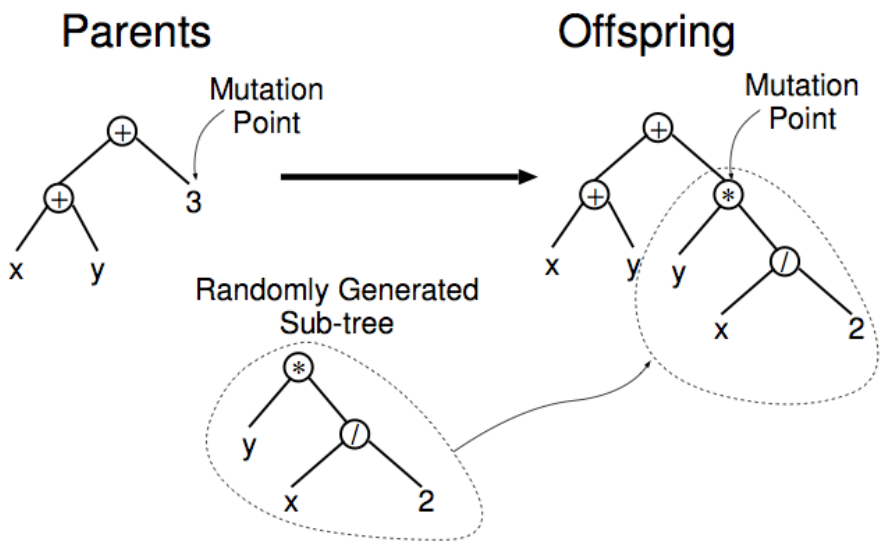
Main evolution loop ('vanilla GP')

```
 1: procedure GeneticProgramming(f, 𝒥)          ▷ f - fitness function, 𝒥 - instruction set
 2:     𝒫 ← {p ← RandomProgram(𝒥)}              ▷ Initialize population
 3:     repeat                                  ▷ Main loop over generations
 4:         for p ∈ 𝒫 do                        ▷ Evaluation
 5:             p.f ← f(p)                      ▷ p.f is a 'field' in program p that stores its fitness
 6:         end for
 7:         𝒫′ ← ∅                              ▷ Next population
 8:         repeat                              ▷ Breeding loop
 9:             p₁ ← TournamentSelection(𝒫)     ▷ First parent
10:             p₂ ← TournamentSelection(𝒫)     ▷ Second parent
11:             (o₁, o₂) ← Crossover(p₁, p₂)
12:             o₁ ← Mutation(o₁, 𝒥)
13:             o₂ ← Mutation(o₂, 𝒥)
14:             𝒫′ ← 𝒫′ ∪ {o₁, o₂}
15:         until |𝒫′| = |𝒫|
16:         𝒫 ← 𝒫′
17:     until StoppingCondition(𝒫)
18:     return arg max_{p∈𝒫} p.f
19: end procedure
```

Search operators: Mutation

Mutation: replace a randomly selected subexpression with a new randomly generated subexpression.

```
1: function Mutation(p, 𝒥)
2:     repeat
3:         s ← Random node in p
4:         s′ ← RandomProgram(𝒥)
5:         p′ ← Replace the subtree rooted in s with s′
6:     until Depth(p′) < d_max                ▷ d_max is the tree depth limit
7:     return p′
8: end function
```

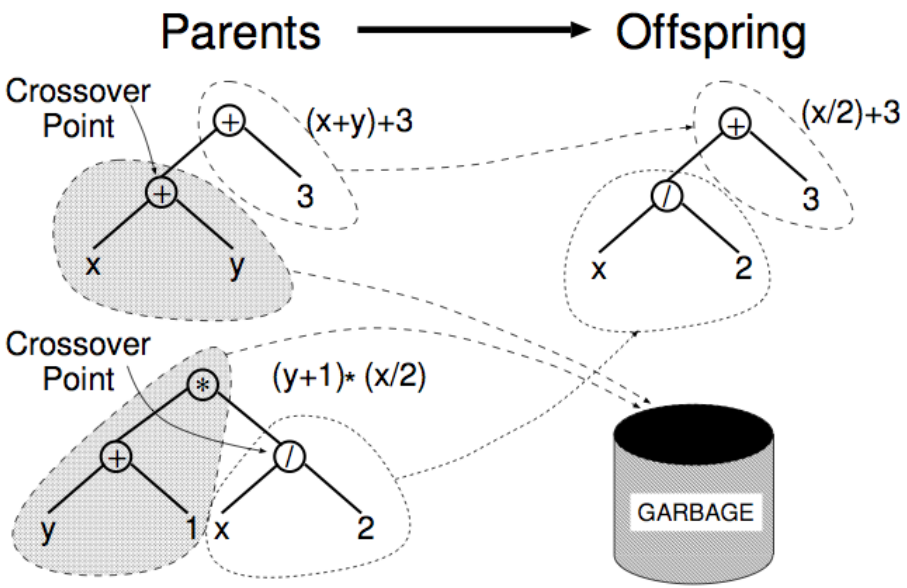


Source: [Poli et al., 2008]

Search operators: Crossover

Crossover: exchange of randomly selected subexpressions (*subtree swapping crossover*).

1: **function** Crossover(p_1, p_2)
2: **repeat**
3: $s_1 \leftarrow$ Random node in p_1
4: $s_2 \leftarrow$ Random node in p_2
5: $(p'_1, p'_2) \leftarrow$ Swap subtrees rooted in s_1 and s_2
6: **until** Depth$(p'_1) < d_{max} \wedge$ Depth$(p'_2) < d_{max}$ ▷ d_{max} is the tree depth limit
7: **return** (p'_1, p'_2)
8: **end function**



Source: [Poli et al., 2008]

Q & A

Q: What is the most likely outcome of application of mutation/crossover to a viable program?

[4] Turns out: In GP, quite many of them can be neutral (*neutral mutations*).

Q & A

Q: What is the most likely outcome of application of mutation/crossover to a viable program?

Hint:

> *But, however many ways there may be of being alive, it is certain that there are vastly more ways of being dead, or rather not alive. (The Blind Watchmaker [Dawkins, 1996])*

A: Most applications of genetic operators are harmful[4]

Yet, GP works. Why?

[4] Turns out: In GP, quite many of them can be neutral (*neutral mutations*).

Q & A

Q: What is the most likely outcome of application of mutation/crossover to a viable program?

Hint:

> *But, however many ways there may be of being alive, it is certain that there are vastly more ways of being dead, or rather not alive. (The Blind Watchmaker [Dawkins, 1996])*

A: Most applications of genetic operators are harmful[4]

Yet, GP works. Why?

> *Mutation is random; natural selection is the very opposite of random (The Blind Watchmaker [Dawkins, 1996])*

[4] Turns out: In GP, quite many of them can be neutral (*neutral mutations*).

Exemplary run: Setup

A mini-run of GP applied to a symbolic regression problem (from: [Poli et al., 2008])

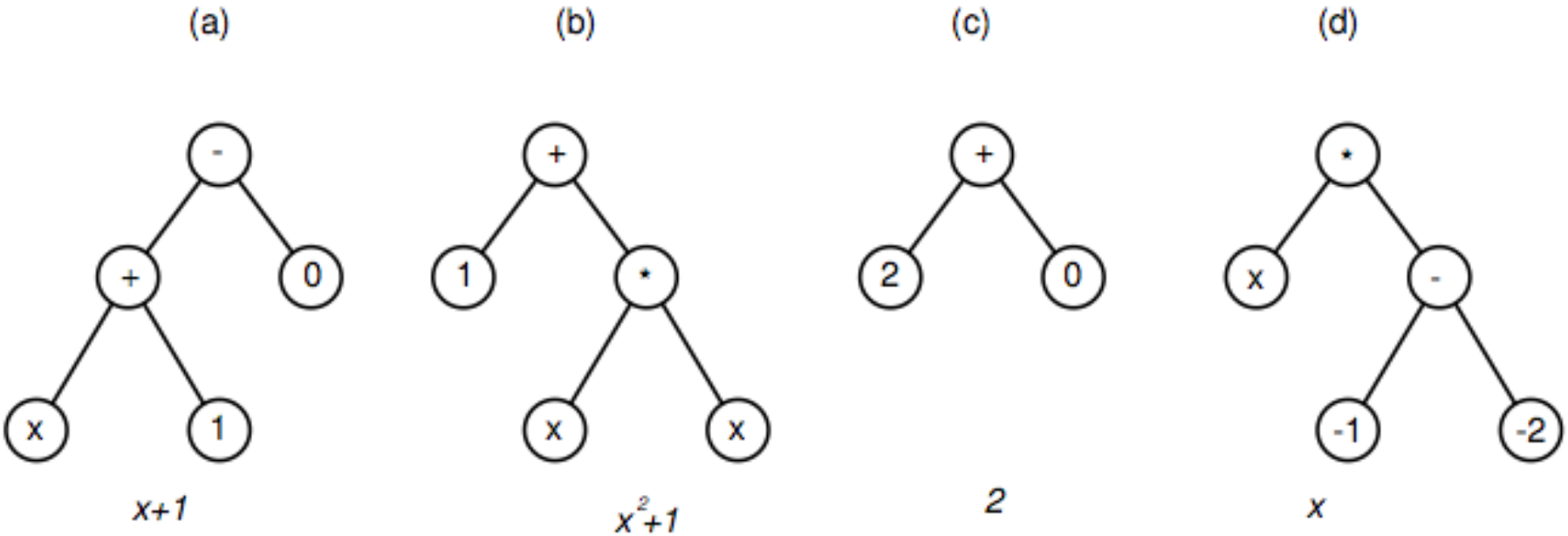
- Objective: Find a program whose output matches $x^2 + x + 1$ over the range $[-1, 1]$.
  - Such tasks can be considered as a form of regression.
  - As solutions are built by manipulating code (symbolic instructions), this is referred to as *symbolic regression*.
- Fitness: sum of absolute errors (City-block distance) for $x \in -1.0, -0.9, ...0.9, 1.0$:

x_i	-1.0	-0.9	...	0	...	0.9	1.0
y_i	1	0.91	...	1	...	2.71	3

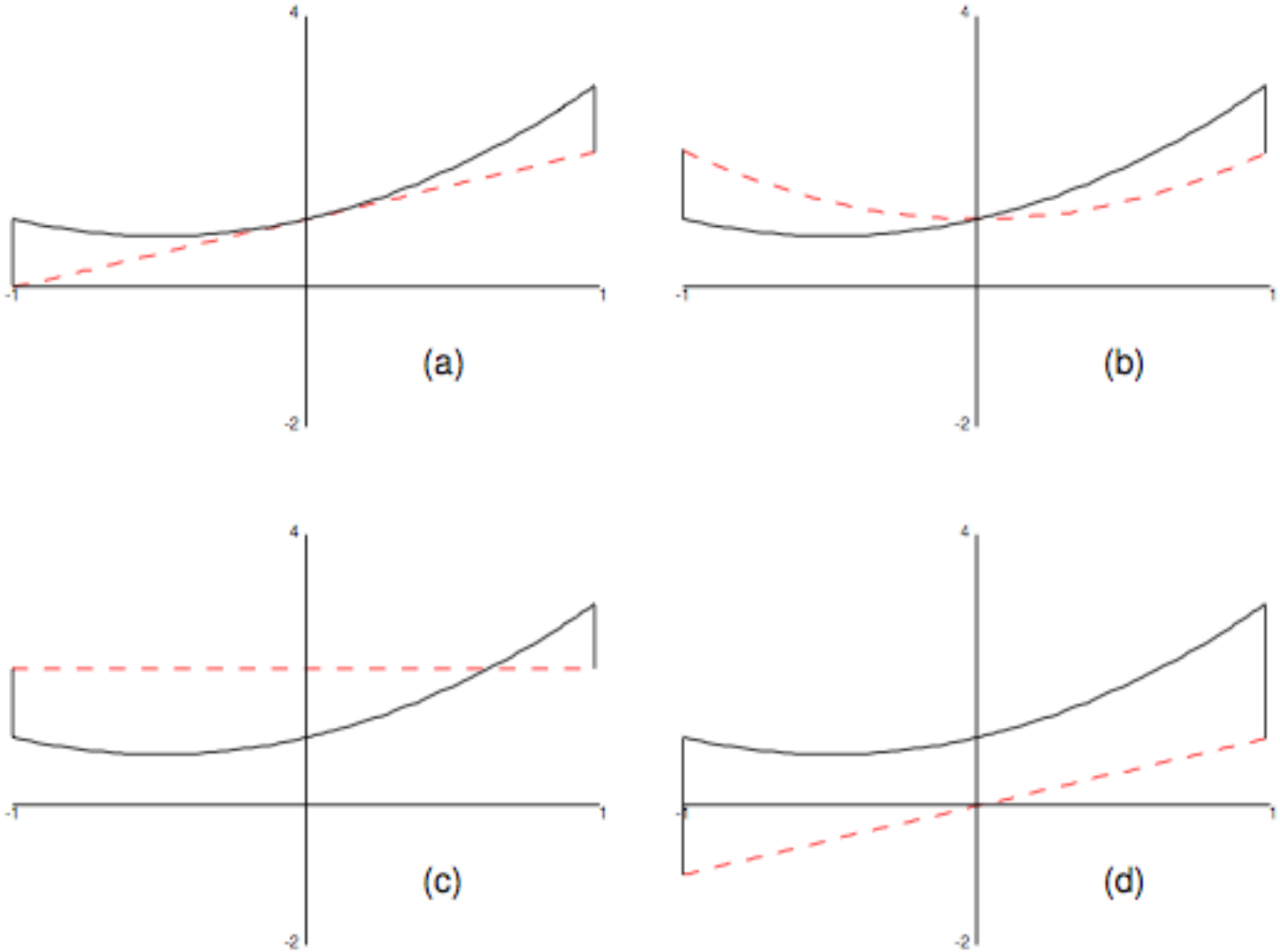
Exemplary run: Setup

- Instruction set:
  - Nonterminal (function) set: +, -, % (protected division), and x; all operating on floats
  - Terminal set: x, and constants chosen randomly between -5 and +5
- Initial population: ramped half-and-half (depth 1 to 2; 50% of terminals are constants)
- Parameters:
  - population size 4,
  - 50% subtree crossover,
  - 25% reproduction,
  - 25% subtree mutation, no tree size limits
- Termination: when an individual with fitness better than 0.1 found
- Selection: fitness proportionate (roulette wheel) non elitist

Initial population (population 0)



Fitness assignment for population 0



Fitness values: f(a)=7.7, f(b)=11.0, f(c)=17.98, f(d)=28.7

Breeding

Assume:

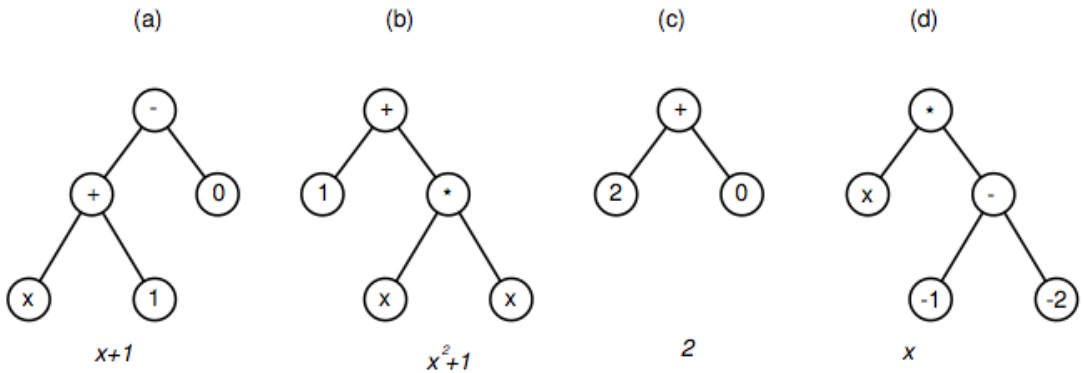
- *a* gets reproduced
- *c* gets mutated (at *locus* 2)
- *a* and *d* get crossed-over
- *a* and *b* get crossed-over

Note:

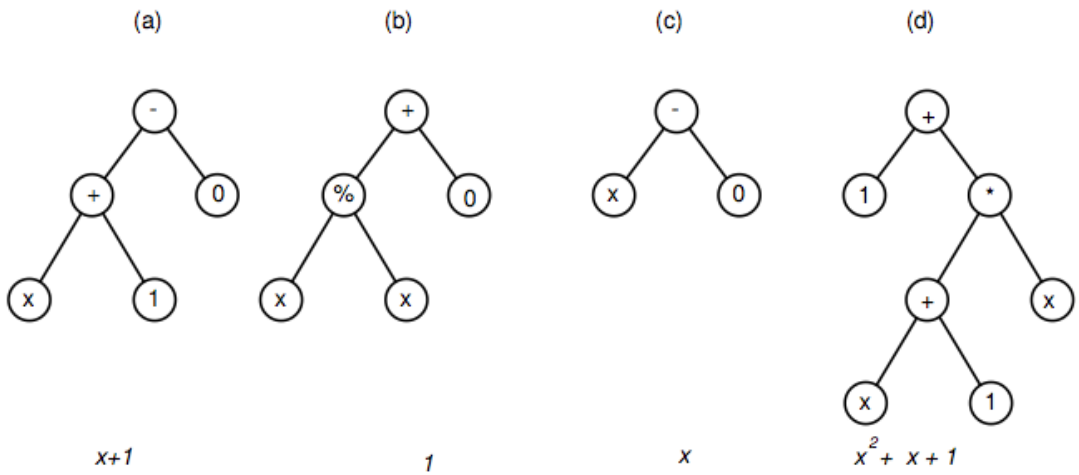
- All parents used; this in general does not have to be the case.

Population 1

Population 0:



Population 1:



Individual *d* in population 1 has fitness 0.

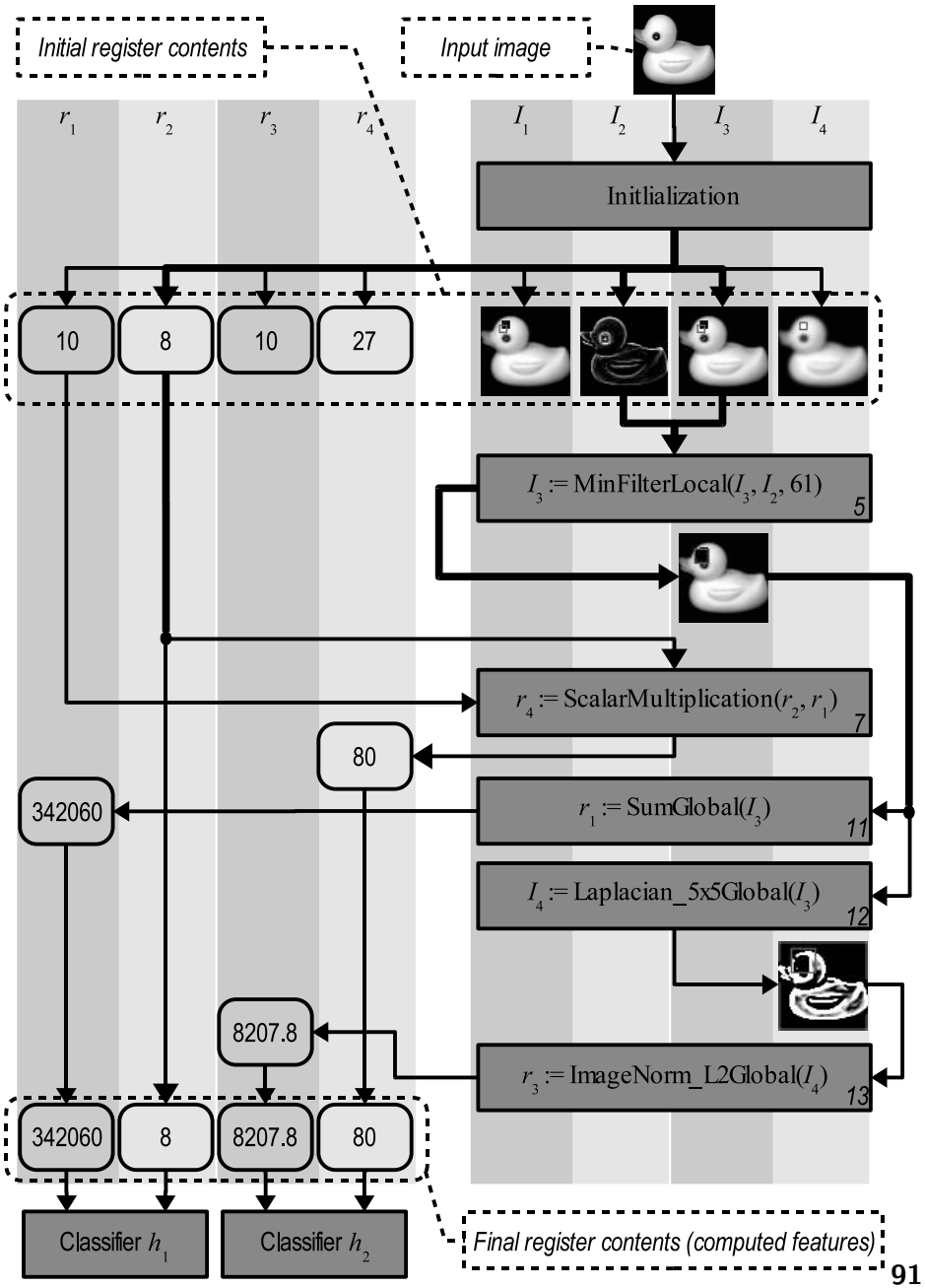
High cost of evaluation

- Running a program on multiple inputs can be expensive.
- Particularly for some types of data, e.g., images

Solutions:

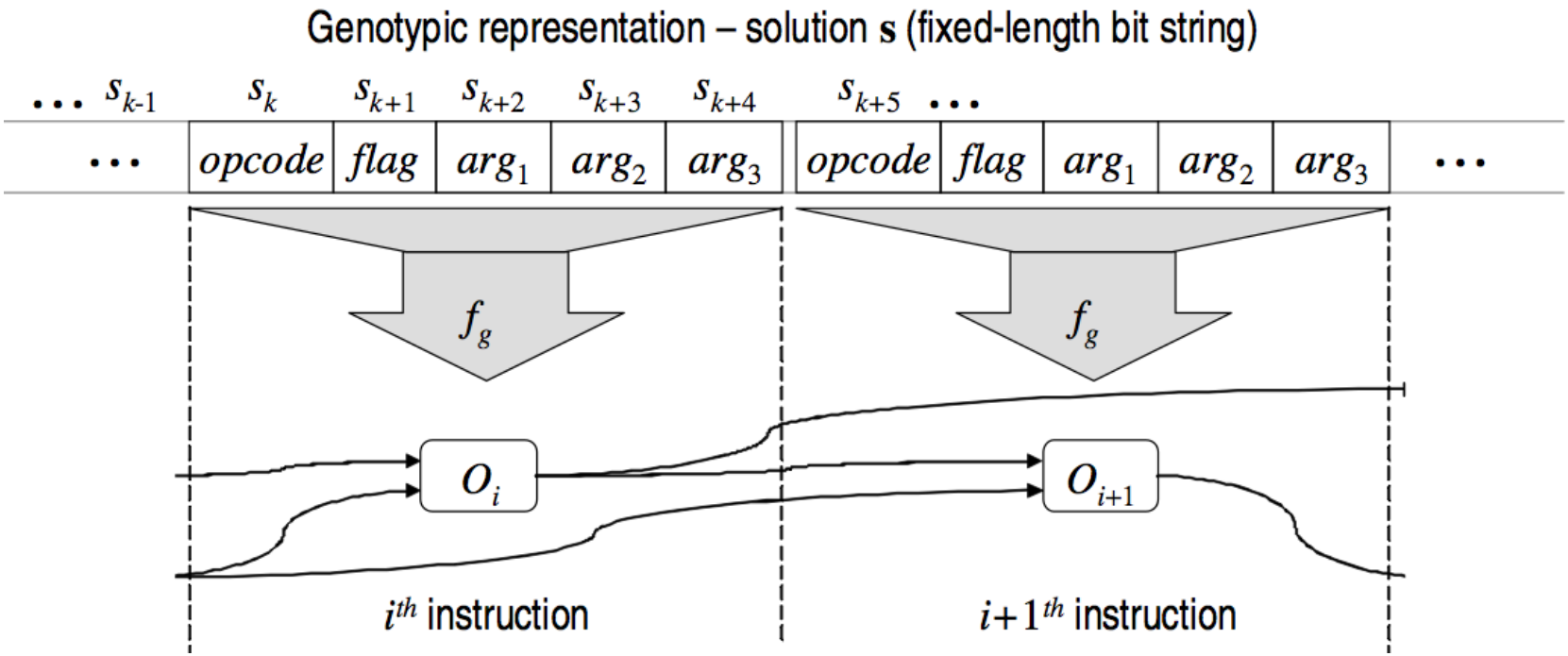
- Caching of outcomes of subprograms
- Parallel execution of programs on particular fitness cases
- Bloat prevention methods

Right: Example from [Krawiec, 2004]. Synthesis of image analysis algorithms, where evaluation by definition incurs high computational cost.

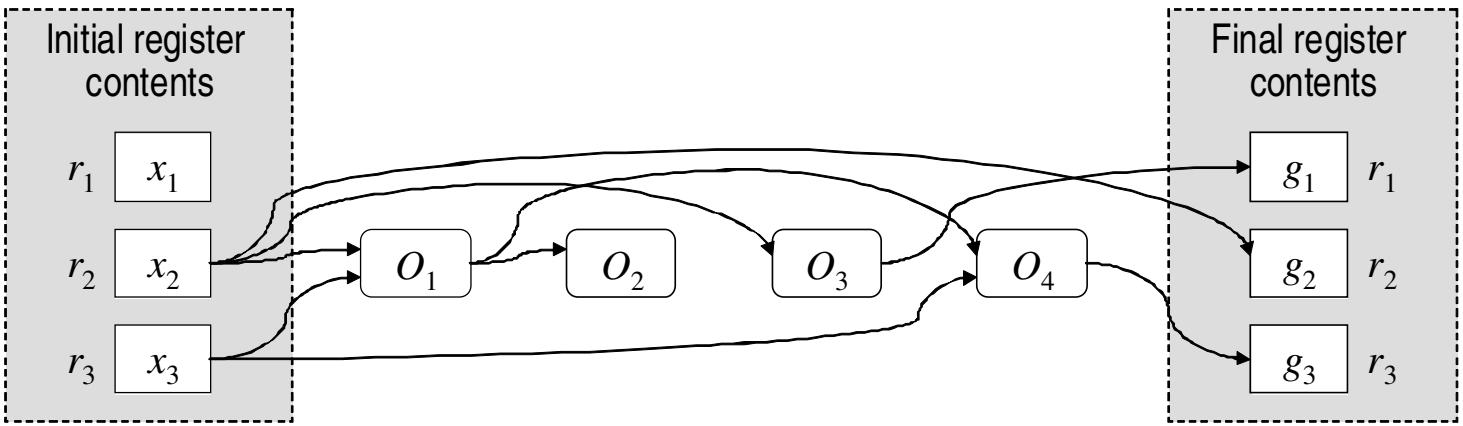


Linear GP

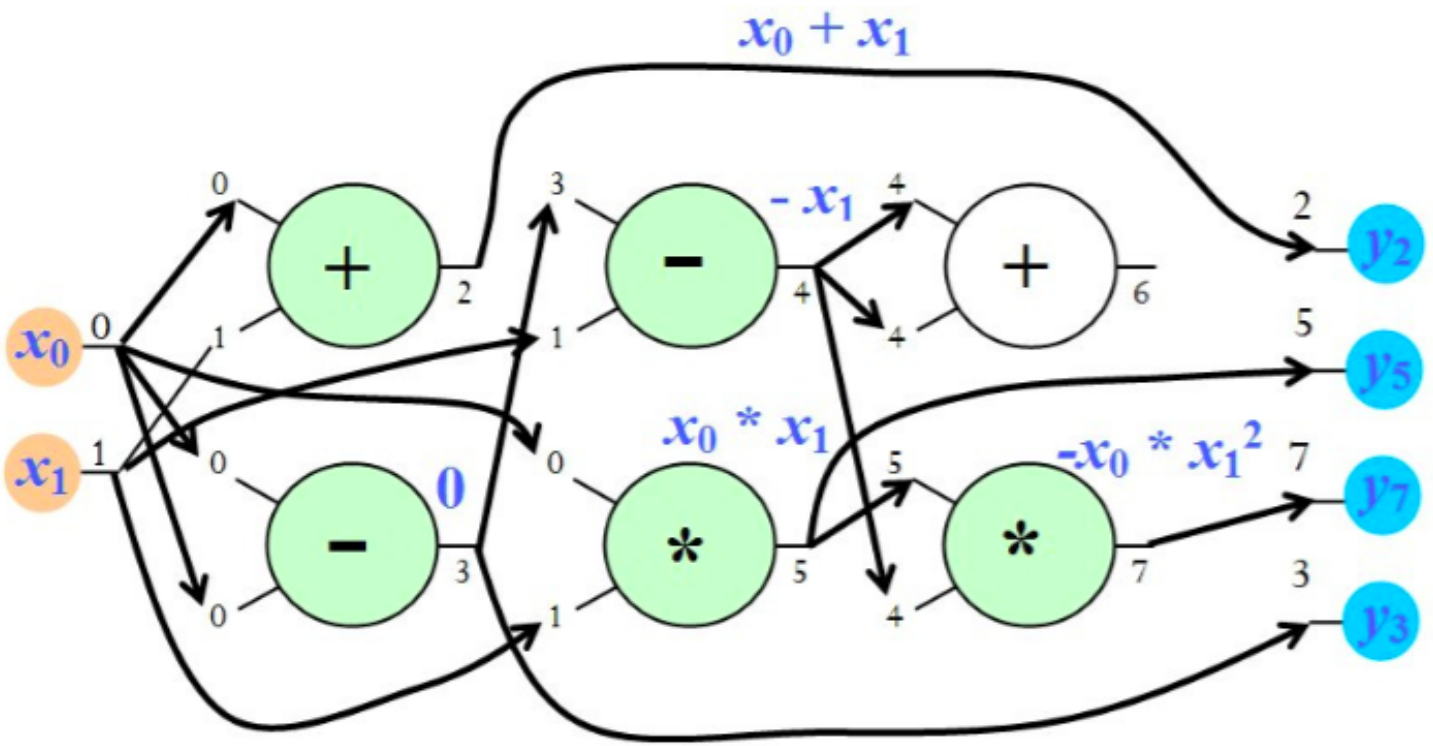
Example from [Krawiec, 2004]: the process of program interpretation:



and the corresponding data flow, including the initial and final register contents:



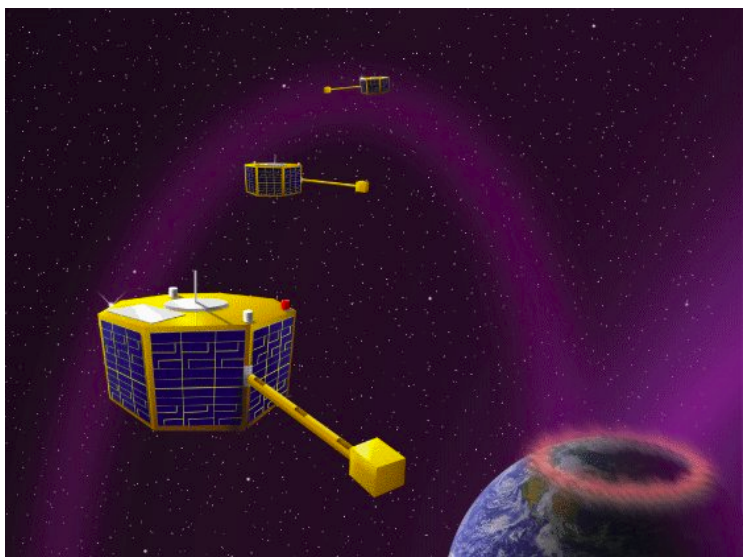
Cartesian GP

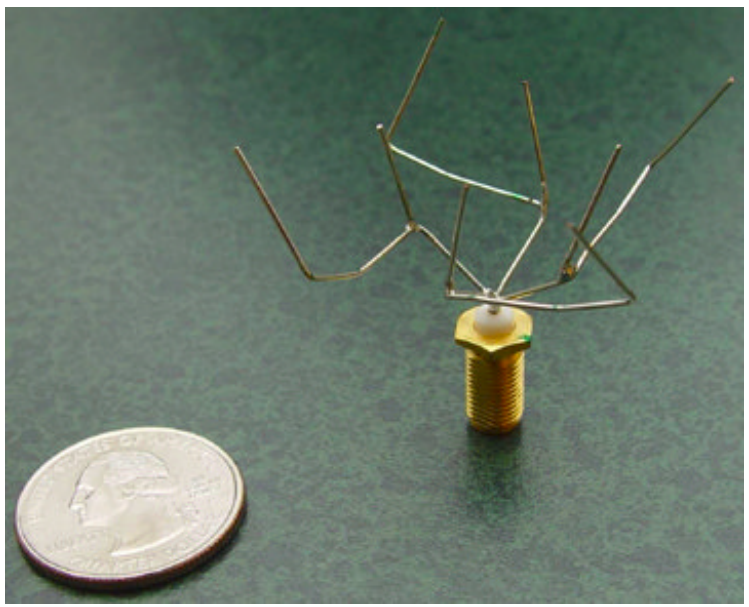


$$y_2 = x_0 + x_1$$
$$y_5 = x_0 * x_1$$
$$y_7 = -x_0 * x_1^2$$
$$y_3 = 0$$

Selected Gold Humies using GP

- 2004: Jason D. Lohn Gregory S. Hornby Derek S. Linden, NASA Ames Research Center,
  An Evolved Antenna for Deployment on NASA's Space Technology 5 Mission





http://idesign.ucsc.edu/papers/hornby_ec11.pdf

Selected Gold Humies using GP

- 2009: S. Forrest, C. Le Goues, ThanhVu Nguyen, W. Weimer
  Automatically finding patches using genetic programming: A Genetic Programming Approach to Automated Software Repair

```
1  void zunebug(int days) {
2    int year = 1980;
3    while (days > 365)  {
4      if (isLeapYear(year)){
5        if (days > 366)  {
6          days -= 366;
7          year += 1;
8        }
9        else {
10       }
11     }
12     else {
13       days -= 365;
14       year += 1;
15     }
16   }
17   printf("current year is %d\n", year);
18 }
```

- Successfully fixes a 'New Year's bug' in Microsoft's MP3 player Zune.

Applications: End user programming

Many end-users need some form of 'programmatic automation' of certain tasks, like commodity traders, graphic designers, chemists, human resource managers, finance pros, ...

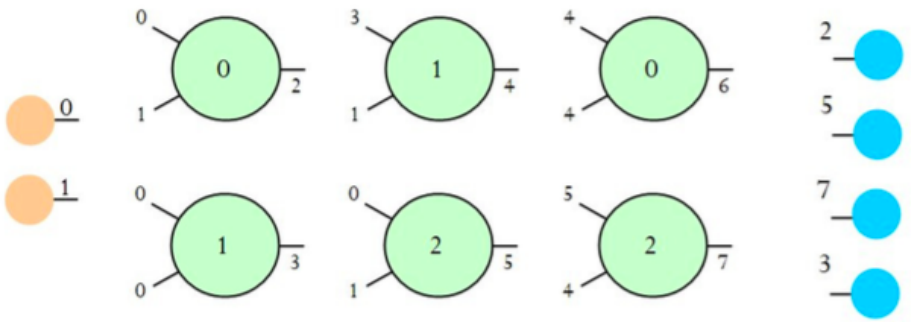
- These users typically lack the technical skills to program from scratch.

General Purpose Programming Assistance

- Synthesis can be used to find tricky/mundane implementation details after human insight has been expressed in the form of a partial program [65]
- Automated Debugging

See also: *Flash fill* [Gulwani et al., 2012]

Example



Function look-up table

Function gene (address)	Action
0	Add
1	Subtract
2	Multiply
3	Divide (protected)

Genotype

0 0 1 1 0 0 1 3 1 2 0 1 0 4 4 2 5 4 2 5 7 3

6.
Uwagi końcowe

Podsumowanie

- Algorytm ewolucyjny = jedna z metaheurystyk.
- Adresowane do trudnych problemów optymalizacji ciągłej, optymalizacji kombinatorycznej i uczenia maszynowego
- Wykazana skuteczność na szerokiej gamie problemów praktycznych.
- Ograniczone, niemniej stale rosnące ugruntowanie teoretyczne.
- Często hybrydyzowany z innymi podejściami.

Metody ewolucyjne vs. metody gradientowe

Metody ewolucyjne

- Iteracyjne
- Stochastyczne
- Przeszukiwanie populacyjne, globalne
- Niskie ryzyko utknięcia w lokalnych optimach
- Dowolność w projektowaniu operatorów przeszukiwania

Metody gradientowe

- Iteracyjne
- Deterministyczne (*)
- Przeszukiwanie lokalne
- Wysokie ryzyko utknięcia w lokalnych optimach
- Operatory przeszukiwania wynikają wprost z reguły spadku gradientu (steepest descent)

Paradygmat ewolucji?

- Paradygmat = “zbiór pojęć i teorii tworzących podstawy danej nauki”

- Paradygmat metaheurystyczny
  - Rozwój algorytmów specjalizowanych do rozwiązywania pewnych (pod)klas problemów, wykorzystujących charakterystykę (strukturę) tych problemów dla zwiększenia efektywności, zarówno w sensie *efficiency*, jak i *efficacy* (*effectiveness*).

- Paradygmat obliczeń ewolucyjnych
  Główne cechy charakterystyczne:
  - Stochastyczny charakter.
  - Wykorzystanie stosunkowo “słabo ukierunkowanych” operatorów przeszukiwania.

Dziękuję za uwagę.