# Biologically-inspired algorithms and models

## 7. Evolutionary design

Maciej Komosinski

# How to represent solutions in ED (evolutionary design)?

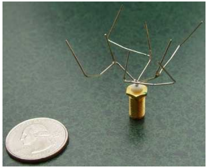


Designs can be passive (static) or active (equipped with actuators–effectors and sometimes also with sensors). One example of ED is therefore evolutionary robotics.

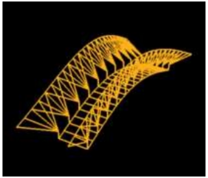
Come up with a few genetic representations for bridge optimization.

# Examples of evolutionary design (1/2)

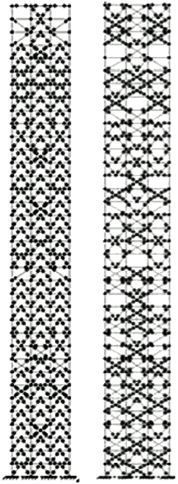




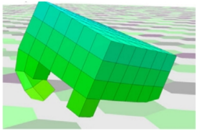
*Automated Antenna Design with Evolutionary Algorithms*, G. Hornby et al., 2006



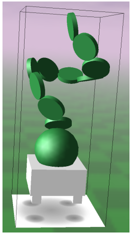
*Combining Structural Analysis and Multi-Objective Criteria for Evolutionary Architectural Design*, J. Byrne et al., 2011



*Evolutionary Design of Steel Structures in Tall Buildings*, R. Kicinger et al., 2005

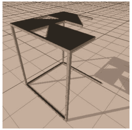


*Evolutionary Developmental Soft Robotics As a Framework to Study Intelligence and Adaptive Behavior in Animals and Plants*, F. Corucci, 2017



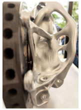
*Framsticks* [KU21a]

[Sim94]



[Hor03]

# Examples of evolutionary design (2/2)

- Early evolved robots
  https://youtu.be/tgJcYx-yewA?t=237
- Wind power plant and turbine optimization
  https://www.youtube.com/watch?v=cNaFhhwTpS8
- Water turbine optimization (what to pay attention to, what are the goals)
  https://youtu.be/fcE6HV1g2kk?t=2477
- Optimizing the aerodynamics of a car
  https://www.youtube.com/watch?v=sw7_XWdd56c&t=25
- Optimizing the structure of a car  
  Czinger 21C: "Using supercomputing and AI (...) the chassis structure is generatively designed. Every component of the structure is pareto optimized for its precise function, not a single gram of material goes to waste."
  https://youtu.be/Pppne2jcgok?t=1541

# Comparison of optimization difficulty

Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Comparison of optimization difficulty

Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Comparison of optimization difficulty



Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Comparison of optimization difficulty



Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| Obvious, natural representation | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Comparison of optimization difficulty

Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| Obvious, natural representation | | |
| Simple definition of neighborhood | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Comparison of optimization difficulty



Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| Obvious, natural representation | | |
| Simple definition of neighborhood | | |
| Many local optima | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Comparison of optimization difficulty

Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| Obvious, natural representation | | |
| Simple definition of neighborhood | | |
| Many local optima | | |
| Strong interactions between parts of the solution | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Comparison of optimization difficulty

Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| Obvious, natural representation | | |
| Simple definition of neighborhood | | |
| Many local optima | | |
| Strong interactions between parts of the solution | | |
| Numerous constraints | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Comparison of optimization difficulty



Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| Obvious, natural representation | | |
| Simple definition of neighborhood | | |
| Many local optima | | |
| Strong interactions between parts of the solution | | |
| Numerous constraints | | |
| Multiple evaluation criteria | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Comparison of optimization difficulty



Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| Obvious, natural representation | | |
| Simple definition of neighborhood | | |
| Many local optima | | |
| Strong interactions between parts of the solution | | |
| Numerous constraints | | |
| Multiple evaluation criteria | | |
| Hard to formalize evaluation criteria | | |
| | | |
| | | |
| | | |
| | | |

# Comparison of optimization difficulty



Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| Obvious, natural representation | | |
| Simple definition of neighborhood | | |
| Many local optima | | |
| Strong interactions between parts of the solution | | |
| Numerous constraints | | |
| Multiple evaluation criteria | | |
| Hard to formalize evaluation criteria | | |
| Deterministic evaluation | | |
| | | |
| | | |
| | | |

# Comparison of optimization difficulty

Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| Obvious, natural representation | | |
| Simple definition of neighborhood | | |
| Many local optima | | |
| Strong interactions between parts of the solution | | |
| Numerous constraints | | |
| Multiple evaluation criteria | | |
| Hard to formalize evaluation criteria | | |
| Deterministic evaluation | | |
| Evaluation includes the aspect of time | | |
| | | |
| | | |

# Comparison of optimization difficulty



Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| Obvious, natural representation | | |
| Simple definition of neighborhood | | |
| Many local optima | | |
| Strong interactions between parts of the solution | | |
| Numerous constraints | | |
| Multiple evaluation criteria | | |
| Hard to formalize evaluation criteria | | |
| Deterministic evaluation | | |
| Evaluation includes the aspect of time | | |
| Evaluation is costly | | |
| | | |

# Comparison of optimization difficulty



Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| Obvious, natural representation | | |
| Simple definition of neighborhood | | |
| Many local optima | | |
| Strong interactions between parts of the solution | | |
| Numerous constraints | | |
| Multiple evaluation criteria | | |
| Hard to formalize evaluation criteria | | |
| Deterministic evaluation | | |
| Evaluation includes the aspect of time | | |
| Evaluation is costly | | |
| Predictable evaluation cost | | |

# Comparison of optimization difficulty

Let's compare the complexity of the classic permutation-based optimization problem and the problem of optimizing designs:

| Property | QAP/TSP | Optimizing designs |
|---|---|---|
| Finite set of solutions | | |
| Discrete-continuous space | | |
| Genotype has constant size | | |
| Obvious, natural representation | | |
| Simple definition of neighborhood | | |
| Many local optima | | |
| Strong interactions between parts of the solution | | |
| Numerous constraints | | |
| Multiple evaluation criteria | | |
| Hard to formalize evaluation criteria | | |
| Deterministic evaluation | | |
| Evaluation includes the aspect of time | | |
| Evaluation is costly | | |
| Predictable evaluation cost | | |
| Easy to estimate similarity | | |

# The level of granularity in evolutionary design

- Conceptual ED: production of high-level conceptual frameworks for designs. New design concepts can be evolved, but building blocks are provided by the designer. Example: a hydropower system as a combination of locations, dam types, tunnel lengths and modes of operation.
- Generative ED: generation of the form of design directly. No pre-defined high-level concepts, no conventions, no imposed knowledge (the Einstellung effect). Low-level building blocks defined. Complex representations. Examples: tables, heatsinks, optical prisms, aerodynamic and hydrodynamic forms, bridges, cranes, EHW, analogue circuits.

# The Einstellung effect and human vs. natural design

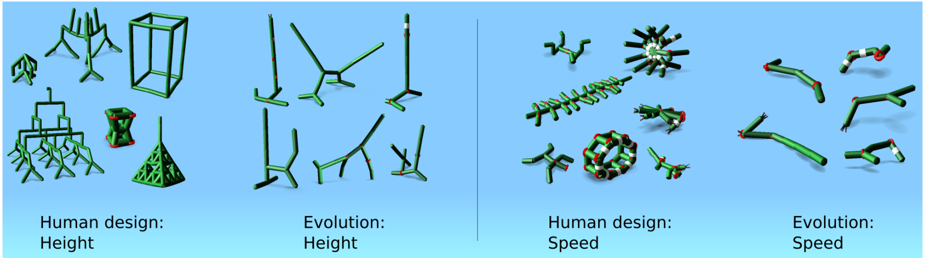
From 7th Int. Conf. on Swarm Intelligence, session on Morphogenetic Engineering:

Engineered products:

- often made of a number of unique, heterogeneous components assembled in a precise and complicated way,
- work deterministically following the specifications given by the designers.

By contrast (compare the figure below), self-organization in natural systems (physical, biological, ecological, social):

- often relies on the repetition of identical agents and stochastic dynamics,
- nontrivial behavior can emerge from relatively simple rules,
- however, most natural patterns can be described with a small number of statistical variables,
- such patterns are random or shaped by boundary conditions, but never exhibit an intrinsic architecture like engineered products do.



# Embryogeny in ED

In evolutionary design, phenotypes are usually much more different from their genotypic representations, than in typical optimization problems. That means that mapping from genotype to phenotype (embryogeny) is needed and may be complex – we talked about it when discussing evolutionary programming.

The goal is good **scalability** (the ability to scale up and create more sophisticated designs [Hor08]) and **evolvability** (the ability to produce offspring that are diverse/more fit [Gaj+19]) – consider the *toothbrush* example [discussion].

# The genotype–phenotype mapping: nature vs. ED

**In nature** embryogeny is defined by interactions between genes, their phenotypic effects and the environment in which the embryo develops. There are chains of interacting 'rules'; the flow of activation is not completely predetermined and preprogrammed; it is dynamic, parallel and adaptive.*

**In evolutionary design** embryogenies can be [Ben99]:

- External (non-evolved). Fixed, static rules, which specify how phenotypes are constructed from the genotypes. E.g. *f0*, *f1*, *fH*, *f7* and *f9* in Framsticks [KU21b].

*https://nautil.us/the-strange-inevitability-of-evolution-235189/

# The genotype–phenotype mapping: nature vs. ED

**In nature** embryogeny is defined by interactions between genes, their phenotypic effects and the environment in which the embryo develops. There are chains of interacting 'rules'; the flow of activation is not completely predetermined and preprogrammed; it is dynamic, parallel and adaptive.*

**In evolutionary design** embryogenies can be [Ben99]:

- External (non-evolved). Fixed, static rules, which specify how phenotypes are constructed from the genotypes. E.g. *f0*, *f1*, *fH*, *f7* and *f9* in Framsticks [KU21b].
- Explicit (evolved). Genotype and embryogeny are evolved simultaneously, but embryogeny is made of pre-defined blocks/features – like iteration, recursion, etc., as in GP (genetic programming). Specialized operators and representations are often needed. E.g. *f4* in Framsticks.

*https://nautil.us/the-strange-inevitability-of-evolution-235189/

# The genotype–phenotype mapping: nature vs. ED

**In nature** embryogeny is defined by interactions between genes, their phenotypic effects and the environment in which the embryo develops. There are chains of interacting 'rules'; the flow of activation is not completely predetermined and preprogrammed; it is dynamic, parallel and adaptive.*

**In evolutionary design** embryogenies can be [Ben99]:

- External (non-evolved). Fixed, static rules, which specify how phenotypes are constructed from the genotypes. E.g. *f0*, *f1*, *fH*, *f7* and *f9* in Framsticks [KU21b].
- Explicit (evolved). Genotype and embryogeny are evolved simultaneously, but embryogeny is made of pre-defined blocks/features – like iteration, recursion, etc., as in GP (genetic programming). Specialized operators and representations are often needed. E.g. *f4* in Framsticks.
- Implicit (evolved). The same genes can be activated and suppressed many times; the same genes can specify *different* functions. Conditional iteration, subroutines, parallel interpretation of genes are allowed. However, it is very difficult to design a good implicit representation. E.g. *fB*, *f6* and *fL* in Framsticks.

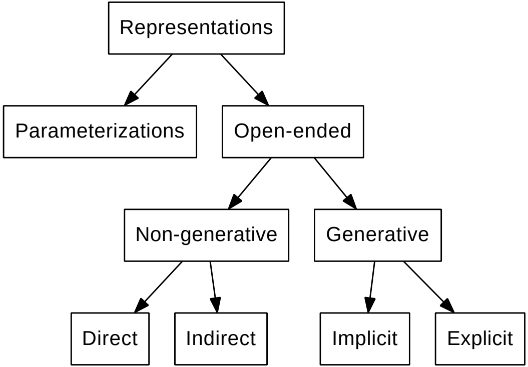
---

*https://nautil.us/the-strange-inevitability-of-evolution-235189/

# The genotype–phenotype mapping: classification



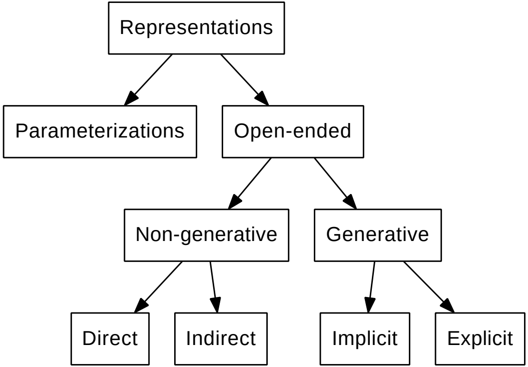
Another, similar classification of embryogenies [Hor03]:




In non-generative representations, each gene is activated once. In Direct and Explicit, the meaning of genes is fixed (not subject to evolution).

# The genotype–phenotype mapping: classification

Another, similar classification of embryogenies [Hor03]:




In non-generative representations, each gene is activated once. In Direct and Explicit, the meaning of genes is fixed (not subject to evolution).

Question: to which category belongs: permutation in TSP, DNA in nature, *f9* in ED?

The development of an efficient embryogeny/mapping may be itself posed as an optimization or machine learning problem ("find an encoding that results in a smooth fitness landscape: maximize FDC" or "find an encoding that makes similar phenotypes genetic neighbors").

Such a problem may be addressed using techniques similar to word embeddings* or (neural) autoencoders** [KKM21].

*https://en.wikipedia.org/wiki/Word_embedding
**https://en.wikipedia.org/wiki/Autoencoder

# References I



[Ben99] Peter Bentley. *Evolutionary design by computers*. Morgan Kaufmann, 1999.

[Gaj+19] Alexander Gajewski et al. "Evolvability ES: scalable and direct optimization of evolvability". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2019, pp. 107–115. URL: https://arxiv.org/pdf/1907.06077.pdf.

[Hor03] Gregory S. Hornby. "Creating complex building blocks through generative representations". In: *Proceedings of the 2003 AAAI Spring Symposium: Computational Synthesis: From Basic Building Blocks to High Level Functionality*. 2003, pp. 98–105. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.323.8779&rep=rep1&type=pdf.

[Hor08] Gregory S. Hornby. "Improving the scalability of generative representations for openended design". In: *Genetic Programming Theory and Practice V* (2008), pp. 125–142.

[KKM21] Piotr Kaszuba, Maciej Komosinski, and Agnieszka Mensfelt. "Automated development of latent representations for optimization of sequences using autoencoders". In: *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2021, pp. 1123–1130. DOI: 10.1109/CEC45853.2021.9504910. URL: http://www.framsticks.com/files/common/LatentRepresentationsForSequencesOptimization.pdf.

[KU21a] Maciej Komosinski and Szymon Ulatowski. *Framsticks website*. 2021. URL: http://www.framsticks.com.

[KU21b] Maciej Komosinski and Szymon Ulatowski. *Genetic representations in Framsticks*. http://www.framsticks.com/a/al_genotype. 2021.

[Sim94] Karl Sims. "Evolving virtual creatures". In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM. 1994, pp. 15–22. URL: https://www.cs.drexel.edu/~david/Classes/Papers/p15-sims.pdf.