Evolutionary Algorithm for Energy System Models

Cruz Enrique Borges Hernández

cruz.borges@deusto.es

Deusto
Universidad de Deusto

WHY

26th of July of 2021

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 891943.

Objectives

- To introduce you to the paradigm of heuristics optimization methods in general and evolutionary algorithm in particular
- To present its main pro and contras
- Learn when Evolutionary Algorithm could provide an added value
- To engineer good Evolutionary Algorithms

Objectives

- To introduce you to the paradigm of heuristics optimization methods in general and evolutionary algorithm in particular
- To present its main pro and contras
- Learn when Evolutionary Algorithm could provide an added value
- To engineer good Evolutionary Algorithms

Objectives

- To introduce you to the paradigm of heuristics optimization methods in general and evolutionary algorithm in particular
- To present its main pro and contras
- Learn when Evolutionary Algorithm could provide an added value
- To engineer good Evolutionary Algorithms

Objectives

- To introduce you to the paradigm of heuristics optimization methods in general and evolutionary algorithm in particular
- To present its main pro and contras
- Learn when Evolutionary Algorithm could provide an added value
- To engineer good Evolutionary Algorithms

Warning

⚠️ I am going to simplify the subject!

"Simpler before in-comprehensible"

Pedro - www.eltamiz.com

Table of Contents

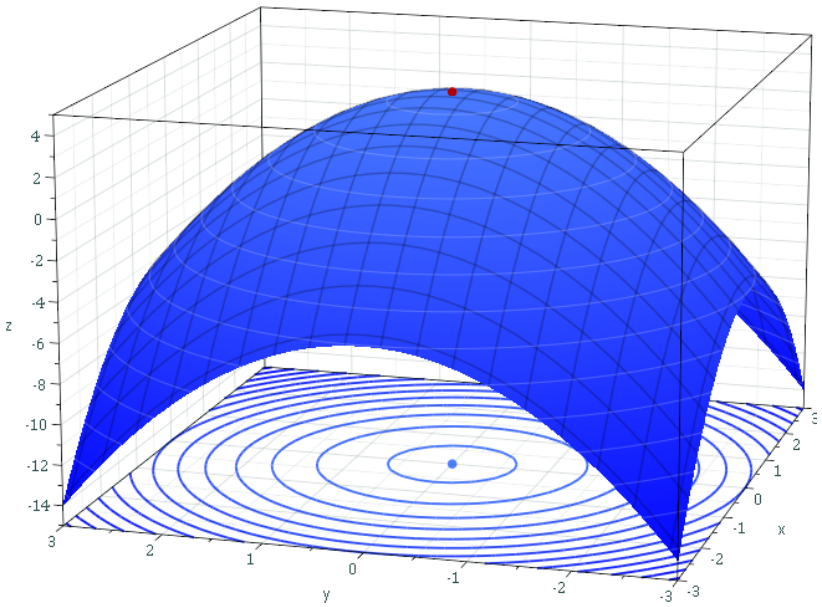
Table of Contents

Table of Contents

Optimization problems

Definition

Given a function $f(\mathbf{x})$, the objective is to found the global minimum that fulfil a set of restrictions $g(\mathbf{x}) = 0$ and $h(\mathbf{x}) \leq 0$



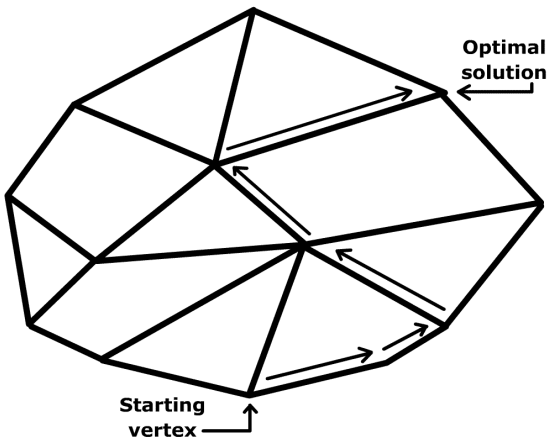
Optimization problem

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g(\mathbf{x}) = 0 \\ & h(\mathbf{x}) \leq 0 \end{aligned}$$

If all the functions are linear…

Linear Programming (Dantzig 1990)

Transform the optimization problem in an iterative procedure that traverse feasible solutions at the edges of the solution space until found the global optimum is found



Victor Treushchenko

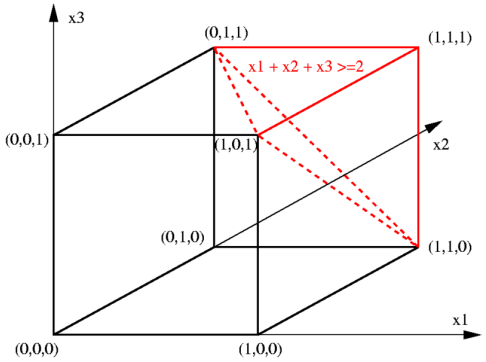
Linear Programming

$$\begin{aligned} \min \quad & \mathbf{c} \cdot \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \end{aligned}$$

If a variable is an integer…

Mixed Integer Linear Programming (Gilmore and Gomory 1963)

Transform the optimization problem in an iterative procedure that cut the feasible space until only the the global optimum is left



Sdo

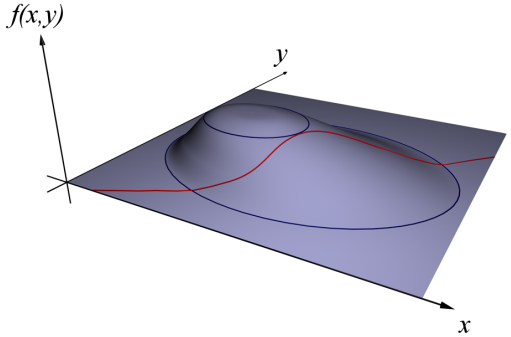
Mixed Integer Linear Programming

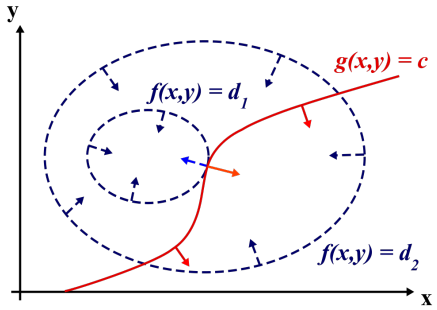
$$\begin{aligned} \min \quad & \mathbf{c} \cdot \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & x_i \in \mathbb{Z} \end{aligned}$$

If a function is non linear (but differentiable)...

Lagrange multiplier (Kuhn 2014)

Transform the optimization problem in the resolution of a non linear equation. Usually, it is needed an numerical method to solve the non-linear equation.





Non-linear optimization

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g(\mathbf{x}) = 0 \end{aligned}$$

$\Longrightarrow$

Lagrange multiplier

$$\mathcal{L}(x, \lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x})$$

When a function is not one of the previous ones?

!?

Evolutionary Algorithm

General idea

Main Characteristics

- Group of algorithms that mimic natural behaviours
- The idea is to have a set of potential solutions and to iteratively evolve them semi-randomly looking for good solutions
- The main elements are:
  - **Candidate:** Representation of a potential solution to the problem
  - **Population:** Set of candidates
  - **Fitness Function:** Function that qualify each solution
  - **Evolutionary Mechanisms:** Methods to modify the Population

Evolutionary Algorithm

General idea

Main Characteristics

- Group of algorithms that mimic natural behaviours
- The idea is to have a set of potential solutions and to iteratively evolve them semi-randomly looking for good solutions
- The main elements are:
  - **Candidate:** Representation of a potential solution to the problem
  - **Population:** Set of candidates
  - **Fitness Function:** Function that qualify each solution
  - **Evolutionary Mechanisms:** Methods to modify the Population

Evolutionary Algorithm

General idea

Main Characteristics

- Group of algorithms that mimic natural behaviours
- The idea is to have a set of potential solutions and to iteratively evolve them semi-randomly looking for good solutions
- The main elements are:
  - **Candidate:** Representation of a potential solution to the problem
  - **Population:** Set of candidates
  - **Fitness Function:** Function that qualify each solution
  - **Evolutionary Mechanisms:** Methods to modify the Population

Evolutionary Algorithm

General idea

Main Characteristics

- Group of algorithms that mimic natural behaviours
- The idea is to have a set of potential solutions and to iteratively evolve them semi-randomly looking for good solutions
- The main elements are:
  - **Candidate:** Representation of a potential solution to the problem
  - **Population:** Set of candidates
  - **Fitness Function:** Function that qualify each solution
  - **Evolutionary Mechanisms:** Methods to modify the Population

Evolutionary Algorithm

General idea

Main Characteristics

- Group of algorithms that mimic natural behaviours
- The idea is to have a set of potential solutions and to iteratively evolve them semi-randomly looking for good solutions
- The main elements are:
  - **Candidate:** Representation of a potential solution to the problem
  - **Population:** Set of candidates
  - **Fitness Function:** Function that qualify each solution
  - **Evolutionary Mechanisms:** Methods to modify the Population

Evolutionary Algorithm

General idea

Main Characteristics

- Group of algorithms that mimic natural behaviours
- The idea is to have a set of potential solutions and to iteratively evolve them semi-randomly looking for good solutions
- The main elements are:
  - **Candidate:** Representation of a potential solution to the problem
  - **Population:** Set of candidates
  - **Fitness Function:** Function that qualify each solution
  - **Evolutionary Mechanisms:** Methods to modify the Population

Evolutionary Algorithm

General idea

Main Characteristics

- Group of algorithms that mimic natural behaviours
- The idea is to have a set of potential solutions and to iteratively evolve them semi-randomly looking for good solutions
- The main elements are:
  - **Candidate:** Representation of a potential solution to the problem
  - **Population:** Set of candidates
  - **Fitness Function:** Function that qualify each solution
  - **Evolutionary Mechanisms:** Methods to modify the Population

Evolutionary Algorithm IV

Theoretical Framework

Holland's schema theorem (Holland 1992)

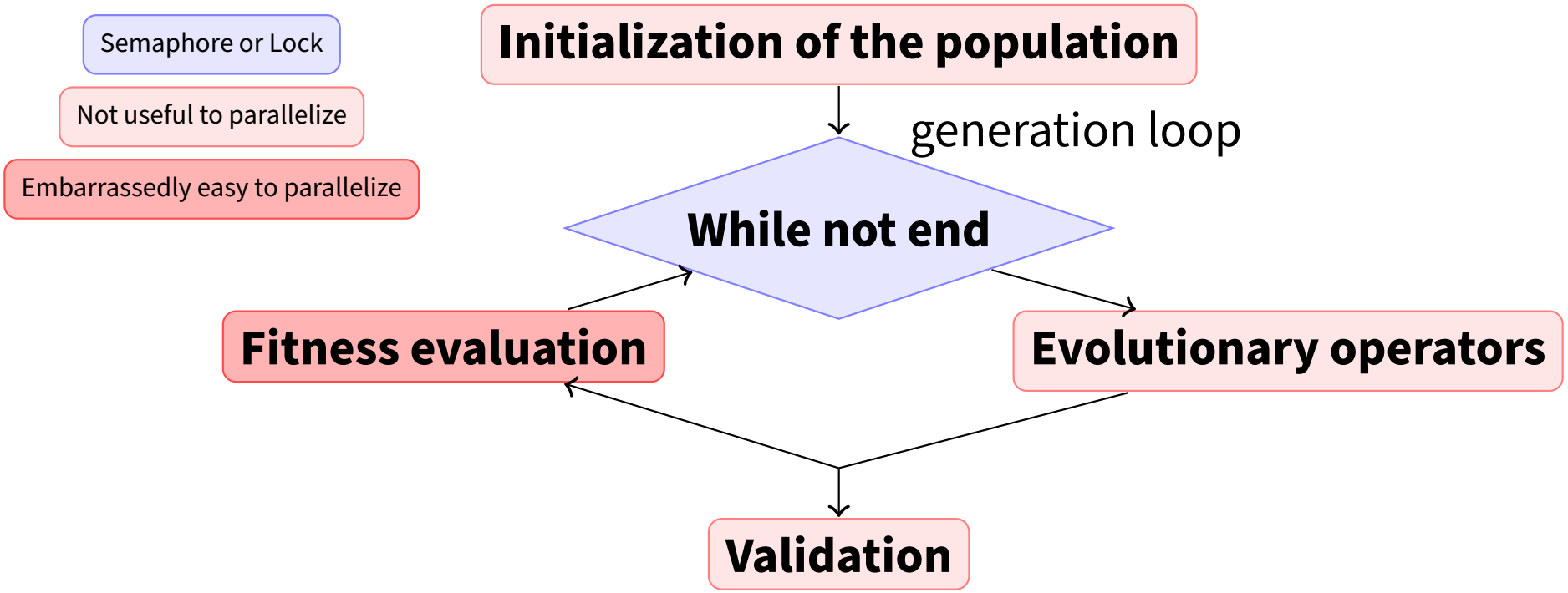
"Short, low-order schemata with above-average fitness increase exponentially in frequency in successive generations"

No free lunch theorem (Wolpert and Macready 1997)

"Any two optimization algorithms are equivalent when their performance is averaged across all possible problems"

Evolutionary Algorithm

Pseudocode



Evolutionary Algorithms

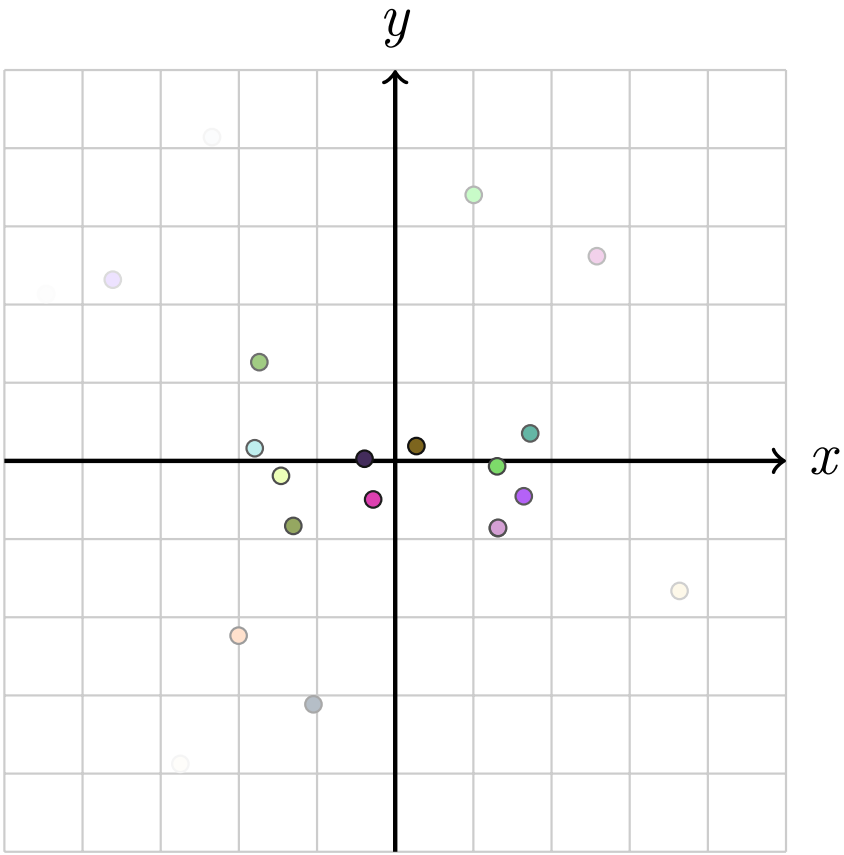
Example

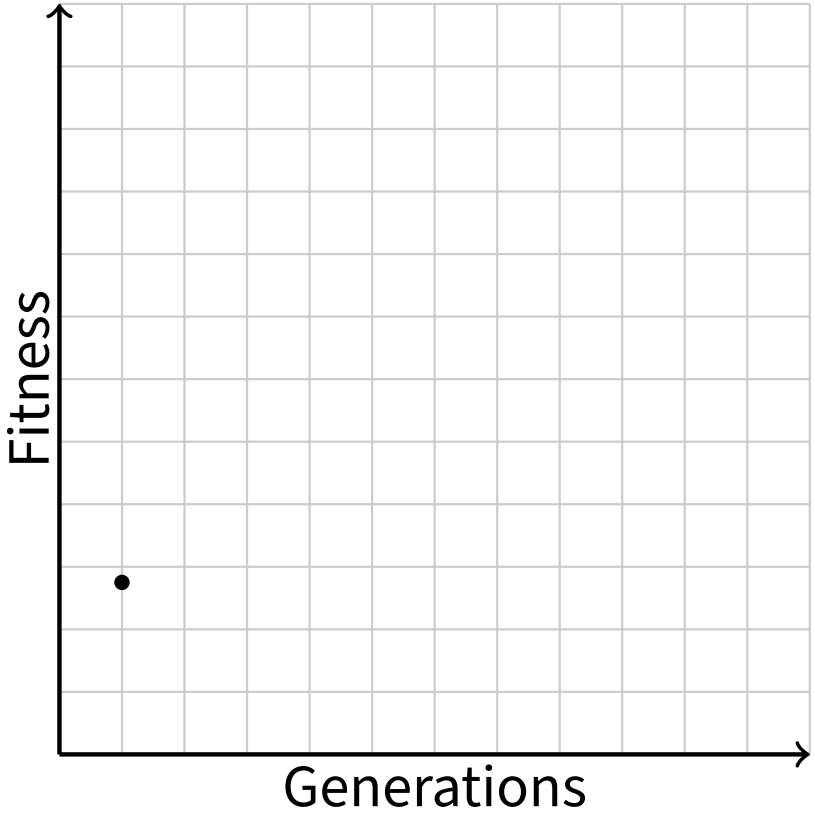
- **Optimization problem:** Minimize $f(x, y) := ||(x, y)|| = \sqrt{x^2 + y^2}$
- **Candidate:** A vector of two real values $X := (x, y) \in \mathbb{R}$
- **Population:** A set of 20 vectors $\{X_1 \dots X_{20}\}$
- **Fitness:** Just the evaluation of $f(x, y)$
- **Genesis:** Create 20 random vectors on $[-5, 5] \times [-5, 5]$
- **Selection:** Simple selection (iteratively over the entire population)
- **Crossover:** Random conic combination $X'' := \lambda X + (1 - \lambda) X'$
- **Mutation:** Adding a random amount λ to one component $X' := (\lambda x, y)$
- **Survival:** The most fitted $f(x, y) < f(x', y')$
- **Stop criteria:** 12 generations
- **Elitism:** No elite

Evolutionary Algorithms

Example: $||f(x, y)||$

1° Generation





Evolutionary Algorithm

Pros and contras

	Traditional	**Evolutionary**
Pros	Convergence	Versatile Easy to implement Highly parallelizable
Contras	Run Time Difficult to implement Constrains on models	Unpredictability

Table of Contents

Look for the right encoding of solutions

Genes

- Evolutionary algorithm are very versatile and could work practically over any combination of variables
- Choose the data structure that best fit your problem (!)
- In a sense, design an evolution algorithm is like building an Object-Oriented program where:
  - `Data_type`: the encoding of a candidate of solution
  - `Methods`: the evolutionary operators
  - `Class`: the population (array of `Data_type`)
  - `Main`: the script to run the algorithm

"All mathematical problems become trivial when you find the right notation"

Rodrigo Camarero Coterillo

Look for the right encoding of solutions

Genes

- Evolutionary algorithm are very versatile and could work practically over any combination of variables
- Choose the data structure that best fit your problem (!)
- In a sense, design an evolution algorithm is like building an Object-Oriented program where:
  - `Data_type`: the encoding of a candidate of solution
  - `Methods`: the evolutionary operators
  - `Class`: the population (array of `Data_type`)
  - `Main`: the script to run the algorithm

"All mathematical problems become trivial when you find the right notation"

Rodrigo Camarero Coterillo

Look for the right encoding of solutions

Genes

- Evolutionary algorithm are very versatile and could work practically over any combination of variables
- Choose the data structure that best fit your problem (!)
- In a sense, design an evolution algorithm is like building an Object-Oriented program where:
  - **`Data type`**: the encoding of a candidate of solution
  - **`Methods`**: the evolutionary operators
  - **`Class`**: the population (array of `Data type`)
  - **`Main`**: the script to run the algorithm

"All mathematical problems become trivial when you find the right notation"

Rodrigo Camarero Coterillo

Look for the right encoding of solutions

Genes

- Evolutionary algorithm are very versatile and could work practically over any combination of variables
- Choose the data structure that best fit your problem (!)
- In a sense, design an evolution algorithm is like building an Object-Oriented program where:
  - **`Data type`**: the encoding of a candidate of solution
  - **`Methods`**: the evolutionary operators
  - **`Class`**: the population (array of `Data type`)
  - **`Main`**: the script to run the algorithm

"All mathematical problems become trivial when you find the right notation"

Rodrigo Camarero Coterillo

Look for the right encoding of solutions

Genes

- Evolutionary algorithm are very versatile and could work practically over any combination of variables
- Choose the data structure that best fit your problem (!)
- In a sense, design an evolution algorithm is like building an Object-Oriented program where:
  - **`Data type`**: the encoding of a candidate of solution
  - **`Methods`**: the evolutionary operators
  - **`Class`**: the population (array of `Data type`)
  - **`Main`**: the script to run the algorithm

"All mathematical problems become trivial when you find the right notation"

Rodrigo Camarero Coterillo

Look for the right encoding of solutions

Genes

- Evolutionary algorithm are very versatile and could work practically over any combination of variables
- Choose the data structure that best fit your problem (!)
- In a sense, design an evolution algorithm is like building an Object-Oriented program where:
  - **`Data type`**: the encoding of a candidate of solution
  - **`Methods`**: the evolutionary operators
  - **`Class`**: the population (array of `Data type`)
  - **`Main`**: the script to run the algorithm

"All mathematical problems become trivial when you find the right notation"

Rodrigo Camarero Coterillo

Look for the right encoding of solutions

Genes

- Evolutionary algorithm are very versatile and could work practically over any combination of variables
- Choose the data structure that best fit your problem (!)
- In a sense, design an evolution algorithm is like building an Object-Oriented program where:
  - **`Data type`**: the encoding of a candidate of solution
  - **`Methods`**: the evolutionary operators
  - **`Class`**: the population (array of `Data type`)
  - **`Main`**: the script to run the algorithm

"All mathematical problems become trivial when you find the right notation"

Rodrigo Camarero Coterillo

Use operators that respect the natural structures

Semantic structures

- Try to identify the semantic structures of your genes
- Evolutionary operators should not destroy these structures (!)
- In fact, what work best is to swap these patters between candidates
- Keep always a random mutator although (!)

Use operators that respect the natural structures

Semantic structures

- Try to identify the semantic structures of your genes
- Evolutionary operators should not destroy these structures (!)
- In fact, what work best is to swap these patters between candidates
- Keep always a random mutator although (!)

Use operators that respect the natural structures

Semantic structures

- Try to identify the semantic structures of your genes
- Evolutionary operators should not destroy these structures (!)
- In fact, what work best is to swap these patters between candidates
- Keep always a random mutator although (!)

Use operators that respect the natural structures

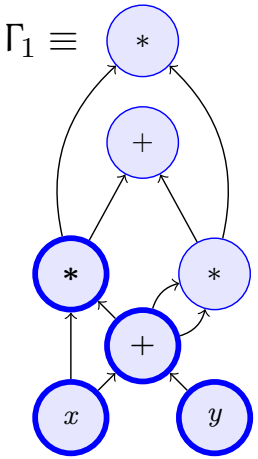
Semantic structures

- Try to identify the semantic structures of your genes
- Evolutionary operators should not destroy these structures (!)
- In fact, what work best is to swap these patters between candidates
- Keep always a random mutator although (!)

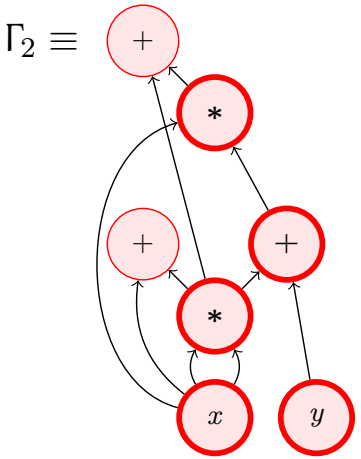
Use operators that respect the natural structures

Semantic structures

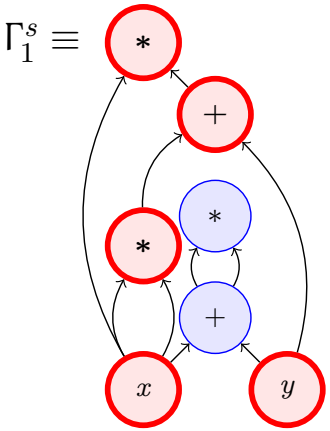
Father 1



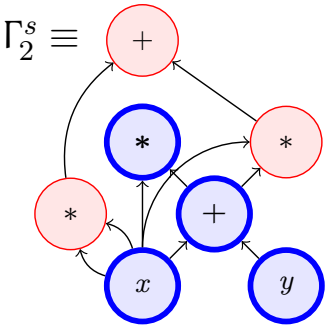
Father 2



Child 1



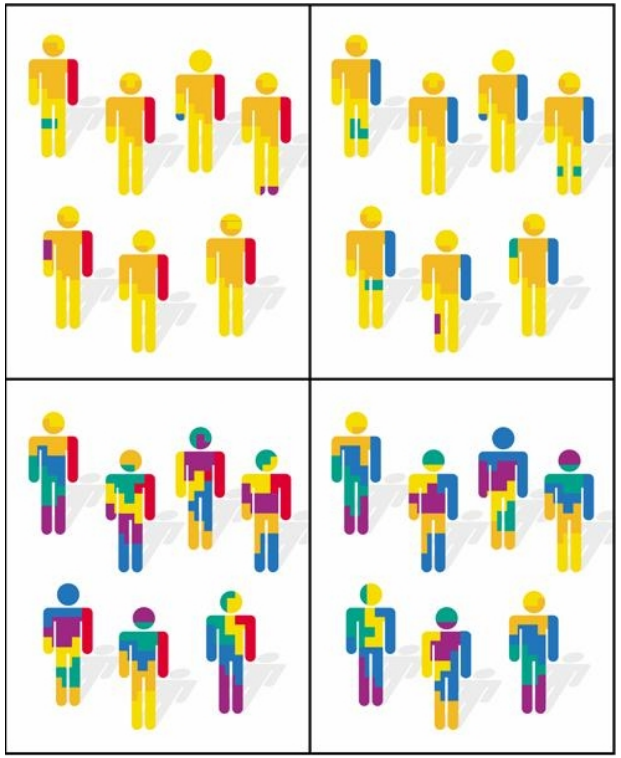
Child 2



Be obsess with the population's diversity

Selection operator

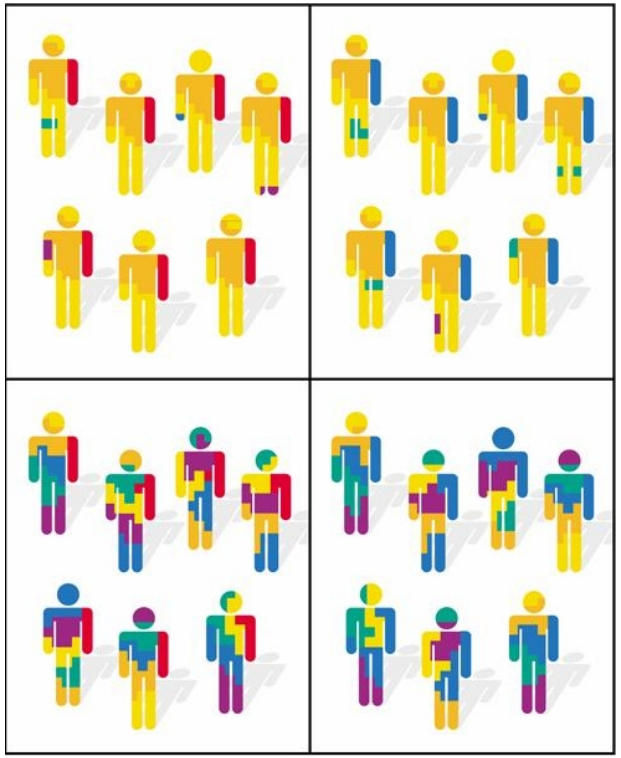
- It is always a good idea to keep the diversity on the population
- Try to not duplicate solutions when operating
- Rank based selection and survival operators



Be obsess with the population's diversity

Selection operator

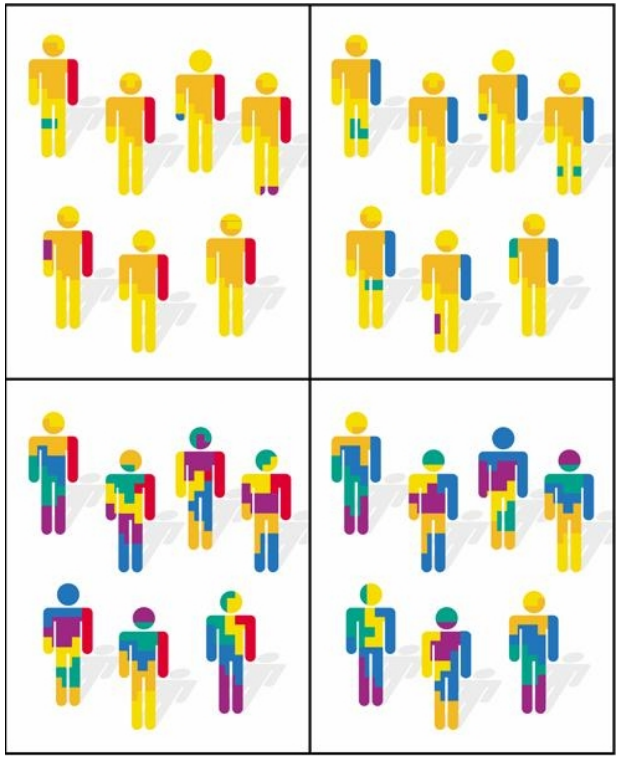
- It is always a good idea to keep the diversity on the population
- Try to not duplicate solutions when operating
- Rank based selection and survival operators



Be obsess with the population's diversity

Selection operator

- It is always a good idea to keep the diversity on the population
- Try to not duplicate solutions when operating
- Rank based selection and survival operators



Choose the right fitness function

Fitness landscape

- The best fitness functions are “hilly”
- Very “flat” fitness functions does not allow the evolutionary operators to prioritize the search space and thus the evolutionary algorithm is just a random search

Choose the right fitness function

Fitness landscape

- The best fitness functions are “hilly”
- Very “flat” fitness functions does not allow the evolutionary operators to prioritize the search space and thus the evolutionary algorithm is just a random search

Expend your time wisely

Fitness function

- If you need to optimize the run time, start with the fitness function
- If you need to squize more time consider using a subrogate function of the fitness function and only evaluate the full fitness every several generations

Expend your time wisely

Fitness function

- If you need to optimize the run time, start with the fitness function
- If you need to squize more time consider using a subrogate function of the fitness function and only evaluate the full fitness every several generations

Keep a healthy amount of randomness

Mutation operator

- Keep always a mutation operator that “randomly” cover the entire search space
- Mutations should be more radical in the first generations and smaller latter on
- This strategy allow you to overcome potential bottlenecks at the first generations keeping the good solutions found at the end
- Beware that this should not be the dominant evolutionary factor (!)
- Beware[2] that this operator should keep as much as possible the underlying semantic structure

Keep a healthy amount of randomness

Mutation operator

- Keep always a mutation operator that “randomly” cover the entire search space
- Mutations should be more radical in the first generations and smaller latter on
- This strategy allow you to overcome potential bottlenecks at the first generations keeping the good solutions found at the end
- Beware that this should not be the dominant evolutionary factor (!)
- Beware[2] that this operator should keep as much as possible the underlying semantic structure

Keep a healthy amount of randomness

Mutation operator

- Keep always a mutation operator that “randomly” cover the entire search space
- Mutations should be more radical in the first generations and smaller latter on
- This strategy allow you to overcome potential bottlenecks at the first generations keeping the good solutions found at the end
- Beware that this should not be the dominant evolutionary factor (!)
- Beware[2] that this operator should keep as much as possible the underlying semantic structure

Keep a healthy amount of randomness

Mutation operator

- Keep always a mutation operator that “randomly” cover the entire search space
- Mutations should be more radical in the first generations and smaller latter on
- This strategy allow you to overcome potential bottlenecks at the first generations keeping the good solutions found at the end
- Beware that this should not be the dominant evolutionary factor (!)
- Beware[2] that this operator should keep as much as possible the underlying semantic structure

Keep a healthy amount of randomness

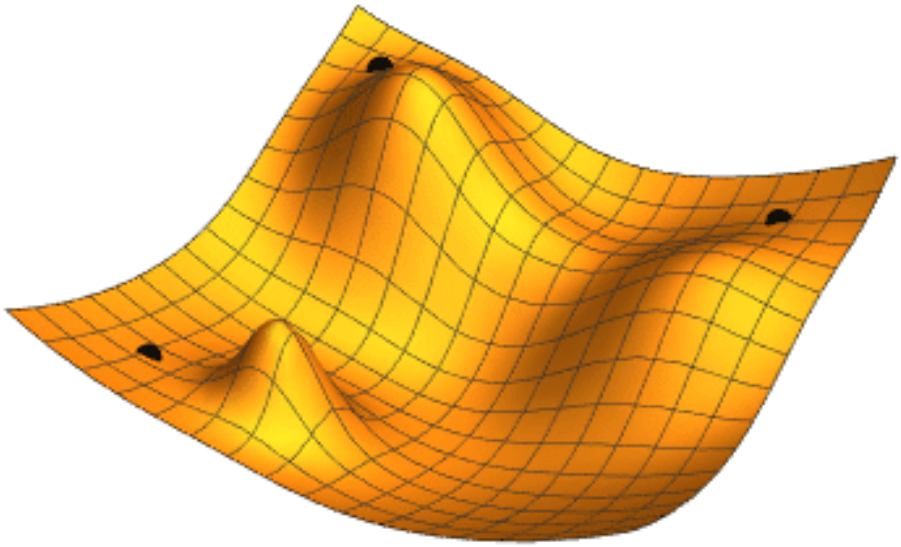
Mutation operator

- Keep always a mutation operator that “randomly” cover the entire search space
- Mutations should be more radical in the first generations and smaller latter on
- This strategy allow you to overcome potential bottlenecks at the first generations keeping the good solutions found at the end
- Beware that this should not be the dominant evolutionary factor (!)
- Beware[2] that this operator should keep as much as possible the underlying semantic structure

Introduce “follow the leader” strategies

Elitism operator

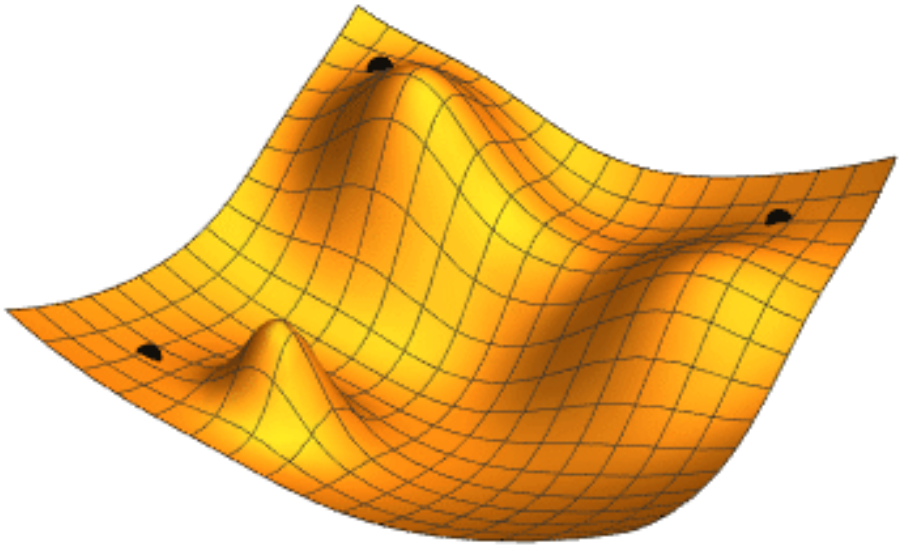
- It is always a good idea to keep the best solution between generation (elitism operator)
- Evolutionary operators that take into consideration the elite tends to reach better solutions faster (Particle Swarm operator and Differential operator)



Introduce “follow the leader” strategies

Elitism operator

- It is always a good idea to keep the best solution between generation (elitism operator)
- Evolutionary operators that take into consideration the elite tends to reach better solutions faster (Particle Swarm operator and Differential operator)



Divide et impera

Co-evolution

- When you have semantic blocks that can be isolated it is a good idea to split the candidates to create two easier problems
- Example:
  - **Block I:** Selection of devices at a building (boolean vector)
  - **Block II:** Hourly set points for each device for a day (vector of 24 floats for each device)
- The best idea is to use the elite element of the other block to calculate the fitness and perform several (more than 1) generations each time for each block
- Beware, there are not silver bullets

Divide et impera

Co-evolution

- When you have semantic blocks that can be isolated it is a good idea to split the candidates to create two easier problems
- Example:
  - **Block I**: Selection of devices at a building (boolean vector)
  - **Block II**: Hourly set points for each device for a day (vector of 24 floats for each device)
- The best idea is to use the elite element of the other block to calculate the fitness and perform several (more than 1) generations each time for each block
- Beware, there are not silver bullets

Divide et impera

Co-evolution

- When you have semantic blocks that can be isolated it is a good idea to split the candidates to create two easier problems
- Example:
  - **Block I**: Selection of devices at a building (boolean vector)
  - **Block II**: Hourly set points for each device for a day (vector of 24 floats for each device)
- The best idea is to use the elite element of the other block to calculate the fitness and perform several (more than 1) generations each time for each block
- Beware, there are not silver bullets

Divide et impera

Co-evolution

- When you have semantic blocks that can be isolated it is a good idea to split the candidates to create two easier problems
- Example:
  - **Block I**: Selection of devices at a building (boolean vector)
  - **Block II**: Hourly set points for each device for a day (vector of 24 floats for each device)
- The best idea is to use the elite element of the other block to calculate the fitness and perform several (more than 1) generations each time for each block
- Beware, there are not silver bullets

Divide et impera

Co-evolution

- When you have semantic blocks that can be isolated it is a good idea to split the candidates to create two easier problems
- Example:
  - **Block I**: Selection of devices at a building (boolean vector)
  - **Block II**: Hourly set points for each device for a day (vector of 24 floats for each device)
- The best idea is to use the elite element of the other block to calculate the fitness and perform several (more than 1) generations each time for each block
- Beware, there are not silver bullets

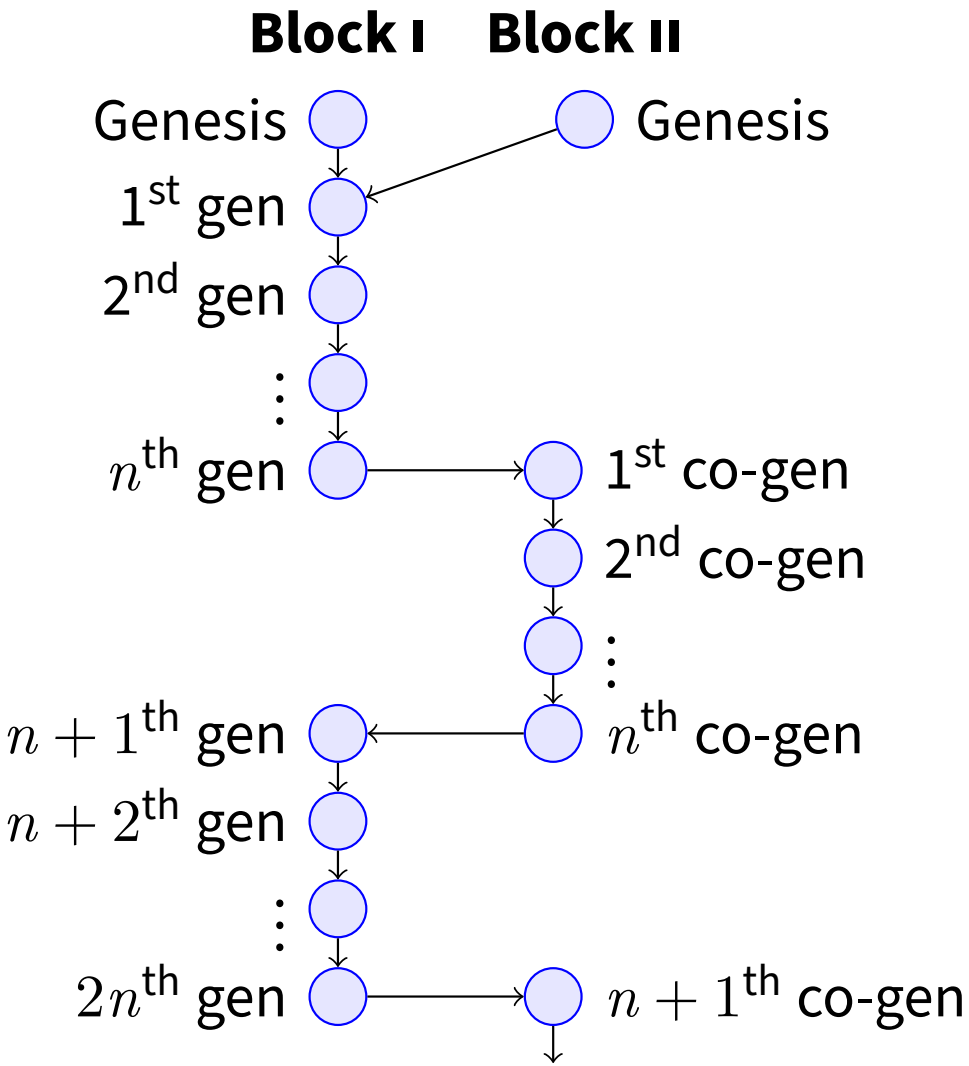
Divide et impera

Co-evolution

- When you have semantic blocks that can be isolated it is a good idea to split the candidates to create two easier problems
- Example:
  - **Block I**: Selection of devices at a building (boolean vector)
  - **Block II**: Hourly set points for each device for a day (vector of 24 floats for each device)
- The best idea is to use the elite element of the other block to calculate the fitness and perform several (more than 1) generations each time for each block
- Beware, there are not silver bullets

Divide et impera

Co-evolution



Polish the end result

Hybrids

- Evolutionary algorithm are good for search but very bad for refine
- It is a good idea to polish the end result using other Local Search techniques
- When feasible, combine some generations (in a co-evolutionary way) of Gradient Descent or Linear Programming to cheaply increase the quality of the results

Polish the end result

Hybrids

- Evolutionary algorithm are good for search but very bad for refine
- It is a good idea to polish the end result using other Local Search techniques
- When feasible, combine some generations (in a co-evolutionary way) of Gradient Descent or Linear Programming to cheaply increase the quality of the results

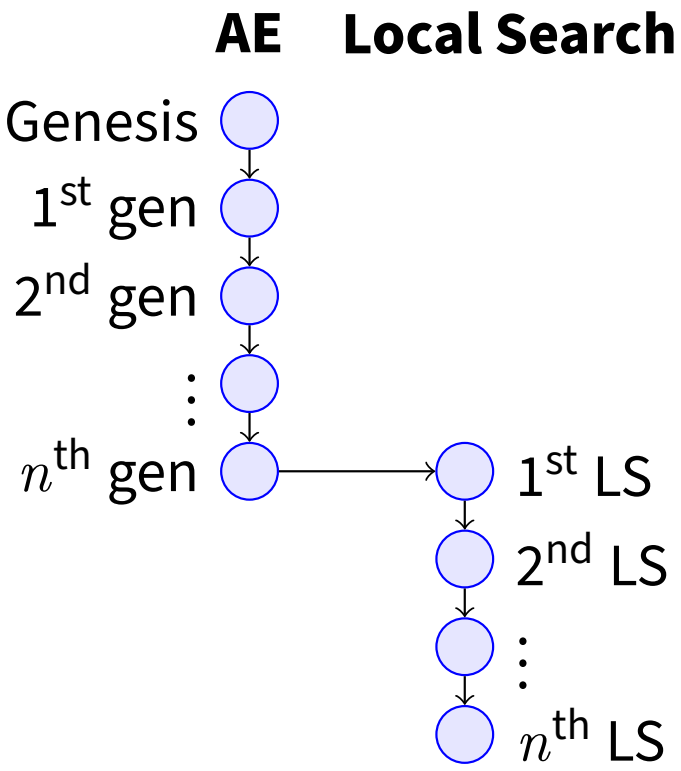
Polish the end result

Hybrids

- Evolutionary algorithm are good for search but very bad for refine
- It is a good idea to polish the end result using other Local Search techniques
- When feasible, combine some generations (in a co-evolutionary way) of Gradient Descent or Linear Programming to cheaply increase the quality of the results

Polish the end result

Hybrids



Exploit the versatility!

Genetic programming

- You are not tight any more particular sets of models or constrains so think out of the box
- Examples:
  - Use simple codification of the solutions without dummy variables
  - Use better non-linear models
  - Evolve a control function using trees, slp (Alonso, Montaña, Puente, and Borges 2009) or neural networks instead of set points for each time step

Exploit the versatility!

Genetic programming

- You are not tight any more particular sets of models or constrains so think out of the box
- Examples:
  - Use simple codification of the solutions without dummy variables
  - Use better non-linear models
  - Evolve a control function using trees, slp (Alonso, Montaña, Puente, and Borges 2009) or neural networks instead of set points for each time step

Exploit the versatility!

Genetic programming

- You are not tight any more particular sets of models or constrains so think out of the box
- Examples:
  - Use simple codification of the solutions without dummy variables
  - Use better non-linear models
  - Evolve a control function using trees, slp (Alonso, Montaña, Puente, and Borges 2009) or neural networks instead of set points for each time step

Exploit the versatility!

Genetic programming

- You are not tight any more particular sets of models or constrains so think out of the box
- Examples:
  - Use simple codification of the solutions without dummy variables
  - Use better non-linear models
  - Evolve a control function using trees, slp (Alonso, Montaña, Puente, and Borges 2009) or neural networks instead of set points for each time step

Exploit the versatility!

Genetic programming

- You are not tight any more particular sets of models or constrains so think out of the box
- Examples:
  - Use simple codification of the solutions without dummy variables
  - Use better non-linear models
  - Evolve a control function using trees, slp (Alonso, Montaña, Puente, and Borges 2009) or neural networks instead of set points for each time step

Evolutionary Algorithm for Energy System Models

Cruz Enrique Borges Hernández

cruz.borges@deusto.es

Deusto
Universidad de Deusto

WHY

26th of July of 2021

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 891943.

Bibliography

- Alonso, Cesar L., José Luis Montaña, Jorge Puente, and Cruz E. Borges (2009). "A new linear genetic programming approach based on straight line programs: Some theoretical and experimental aspects". In: *International Journal on Artificial Intelligence Tools* 18.05, pp. 757–781.
- Dantzig, George B. (1990). "Origins of the Simplex Method". In: *A History of Scientific Computing*. New York, NY, USA: Association for Computing Machinery, pp. 141–151.
- Eiben, AE and JE Smith (2015). *Introduction to Evolutionary Computing Second Edition*. Springer.
- Gilmore, P. C. and R. E. Gomory (1963). "A Linear Programming Approach to the Cutting Stock Problem-Part II". In: *Operations Research* 11.6, pp. 863–888.
- Holland, J.H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. A Bradford book. MIT Press.
- Kuhn, Harold W. (2014). "Nonlinear Programming: A Historical View". In: *Traces and Emergence of Nonlinear Programming*. Ed. by Giorgio Giorgi and Tinne Hoff Kjeldsen. Basel: Springer Basel, pp. 393–414.
- Wolpert, D.H. and W.G. Macready (1997). "No free lunch theorems for optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1, pp. 67–82.