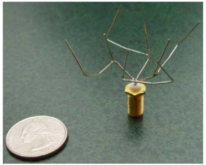
Automated Design Competition at GECCO 2024

Maciej Komosinski Agnieszka Mensfelt Konrad Miazga

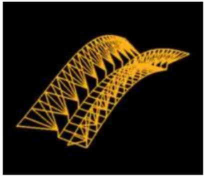
www.framsticks.com

Automated design

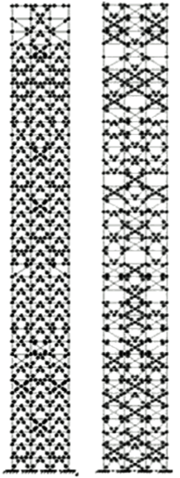
Examples of evolutionary design



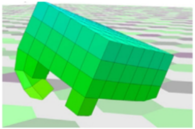
Automated Antenna Design with Evolutionary Algorithms, G. Hornby et al., 2006



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

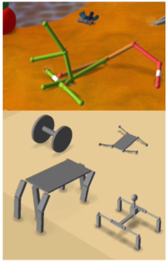


Evolutionary Design of Steel Structures in Tall Buildings, R. Kicinger et al., 2005

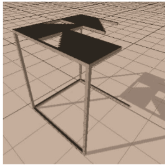


Evolutionary Developmental Soft Robotics As a Framework to Study Intelligence and Adaptive Behavior in Animals and Plants, F. Corucci, 2017

Evolving virtual creatures, K. Sims [Sim94]



Framsticks [KU24]



Generative representations, G. Hornby [Hor03]

Challenges in automated design

- Mixed representations (discrete and continuous)
- Genotypes of variable size
- Non-obvious representation
- Complex genetic operators
- Complex evaluation criteria
- Computationally costly evaluation
- Nondeterministic evaluation

Framsticks

Framsticks – general information

- https://youtu.be/CrWj_l-UrN4?t=60
- https://youtu.be/r5RfTmx3S4g

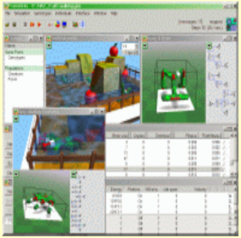
Framsticks – general information

- https://youtu.be/CrWj_l-UrN4?t=60
- https://youtu.be/r5RfTmx3S4g

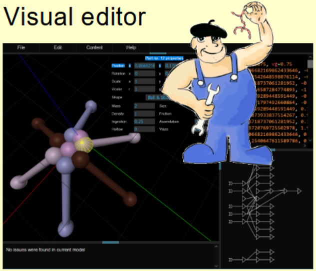
- Developed since 1996
- Authors and main developers: Maciej Komosinski and Szymon Ulatowski
- Volunteers involved in development, experiments, and technical support

Software

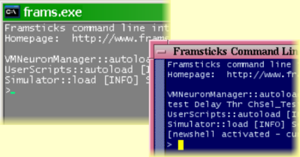
Simulator GUI



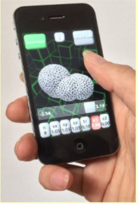
Visual editor



Simulator command-line and network server



Artificial Life (mobile app)

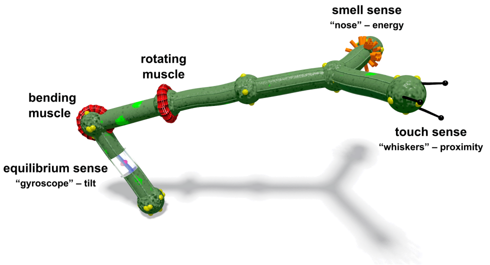


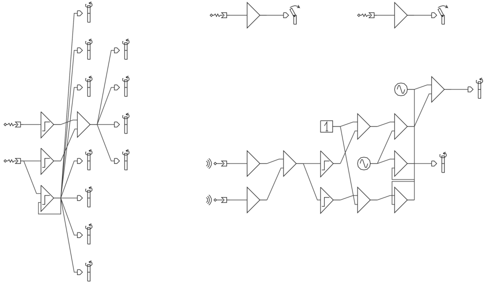
Native library with C++ and Python bindings

```
class FramsticksLib:
    def getSimplest(genetic_format) → str
    def evaluate(genotype_list: list[str]) → list[dict]
    def mutate(genotype_list: list[str]) → list[str]
    def crossOver(geno_parent1: str, geno_parent2: str) → str
    def dissimilarity(genotype_list: list[str]) → np.ndarray
    def isValid(genotype_list: list[str]) → list[bool]
```

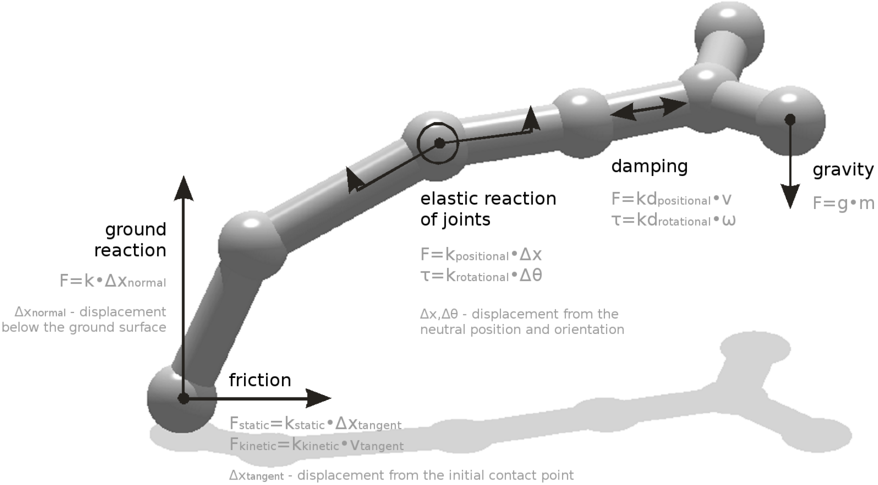
Body and brain

- Composed of "body" and "brain"
- Body made of basic mechanical elements
- Brain made of artificial neurons
- Receptors and effectors: environment ↔ body ↔ brain
- Can be simplified or customized as needed





Simulation – the “MechaStick” engine



Genetics

- Various genetic encodings available
- Custom genetic code can be implemented with its own characteristics, biases, mutation and crossover
- For a new encoding, need to implement genotype $\rightarrow$ phenotype mapping

Genetics – **f0** representation

- All elements directly described
- Basic, internal format
- "Serialization" of a Model
- Supports geometric relativity

Genetics – f0 representation

- All elements directly described
- Basic, internal format
- "Serialization" of a Model
- Supports geometric relativity

```
//0
p:
p:1.0
p:1.5,-0.612,0.612
p:1.5,0.612,-0.612
j:0,1,rx=-0.7854,dx=1.0,0.0,0.0
j:1,2,rx=-0.5184,rz=-1.0472,dx=1.0,0.0,0.0
j:1,3,rx=-0.5184,rz=1.0472,dx=1.0,0.0,0.0
n:j=1,d=@:p=0.25
n:p=3,d=Sin
c:0,1
```

Genetics – **f0** representation

- All elements directly described
- Basic, internal format
- "Serialization" of a Model
- Supports geometric relativity

```
//0
p:
p:1.0
p:1.5,−0.612,0.612
p:1.5,0.612,−0.612
j:0,1,rx=−0.7854,dx=1.0,0.0,0.0
j:1,2,rx=−0.5184,rz=−1.0472,dx=1.0,0.0,0.0
j:1,3,rx=−0.5184,rz=1.0472,dx=1.0,0.0,0.0
n:j=1,d=@:p=0.25
n:p=3,d=Sin
c:0,1
```

Equivalent to this **f1** genotype:

qX(X[@,1:1],X[Sin])

which was converted to **f0** according to the genetic encoding conversion graph.

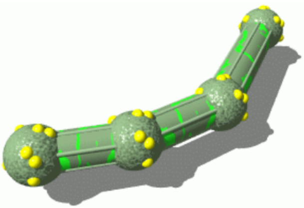
Genetics – f0 genotype–phenotype relation

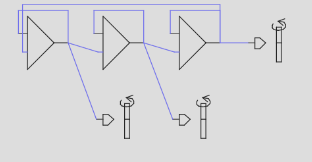

```
//0
p:fr=0.025, vg=0.875
p:0.351, fr=0.025, vg=0.875
p:0.245, 0.324, fr=0.0062, vg=0.875
p:–0.195, 0.397, fr=0.1, vg=0.875
...
j:0, 1, dx=0.351, 0.0, 0.0
j:1, 2, rz=1.884, dx=0.341, 0.0, 0.0
j:1, 3, rz=2.513, dx=0.675, 0.0, 0.0
j:3, 4, rx=0.785, rz=–1.5, dx=0.393, 0.0, 0.0
...
n:j=2, d=@:p=0.625
n:p=4, d=N:in=0.0
n:j=3, d="|:p=0.55,r=0.333333"
...
c:0, 2, 1.272
c:1, 0
c:2, 8, 0.931
...
```

Genetics – f0 genotype–phenotype relation


```
//0
parts
p:fr=0.025, vg=0.875
p:0.351, fr=0.025, vg=0.875
p:0.245, 0.324, fr=0.0062, vg=0.875
p:–0.195, 0.397, fr=0.1, vg=0.875
...
joints
j:0, 1, dx=0.351, 0.0, 0.0
j:1, 2, rz=1.884, dx=0.341, 0.0, 0.0
j:1, 3, rz=2.513, dx=0.675, 0.0, 0.0
j:3, 4, rx=0.785, rz=–1.5, dx=0.393, 0.0, 0.0
...
neurons
n:j=2, d=@:p=0.625
n:p=4, d=N:in=0.0
n:j=3, d="|:p=0.55,r=0.333333"
...
conn's
c:0, 2, 1.272
c:1, 0
c:2, 8, 0.931
...
```

Genetics – f0 genotype–phenotype relation

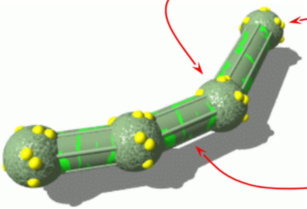


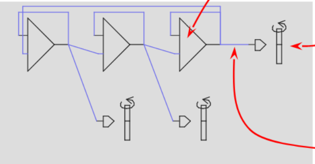


parts
joints
neurons
conn's

```
//0
p:fr=0.025, vg=0.875
p:0.351, fr=0.025, vg=0.875
p:0.245, 0.324, fr=0.0062, vg=0.875
p:–0.195, 0.397, fr=0.1, vg=0.875
...
j:0, 1, dx=0.351, 0.0, 0.0
j:1, 2, rz=1.884, dx=0.341, 0.0, 0.0
j:1, 3, rz=2.513, dx=0.675, 0.0, 0.0
j:3, 4, rx=0.785, rz=–1.5, dx=0.393, 0.0, 0.0
...
n:j=2, d=@:p=0.625
n:p=4, d=N:in=0.0
n:j=3, d="|:p=0.55,r=0.333333"
...
c:0, 2, 1.272
c:1, 0
c:2, 8, 0.931
...
```

Genetics – **f0** genotype–phenotype relation





```
//0
```

parts

```
p:fr=0.025, vg=0.875
p:0.351, fr=0.025, vg=0.875
p:0.245, 0.324, fr=0.0062, vg=0.875
p:–0.195, 0.397, fr=0.1, vg=0.875
...
```

joints

```
j:0, 1, dx=0.351, 0.0, 0.0
j:1, 2, rz=1.884, dx=0.341, 0.0, 0.0
j:1, 3, rz=2.513, dx=0.675, 0.0, 0.0
j:3, 4, rx=0.785, rz=–1.5, dx=0.393, 0.0, 0.0
...
```

neurons

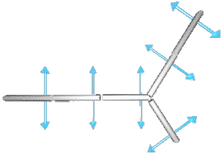
```
n:j=2, d=@:p=0.625
n:p=4, d=N:in=0.0
n:j=3, d="|:p=0.55,r=0.333333"
...
```

conn's

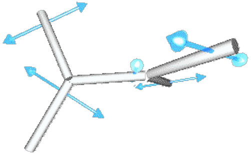
```
c:0, 2, 1.272
c:1, 0
c:2, 8, 0.931
...
```

- Properties are local, relative
- Properties propagate along the body
- Control elements (neurons, sensors) are near elements under control (muscles, sticks)
- Recursive body (tree)
- Any topology of NN
- Human-friendly

XXX(XX,X)

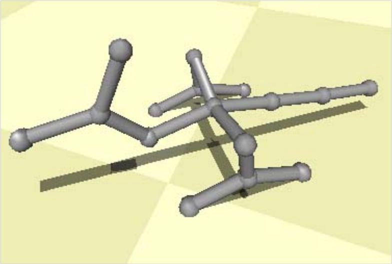


X(X,RR**X(X,X))**

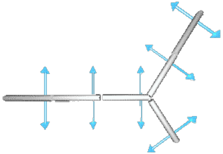


Genetics – **f1** representation

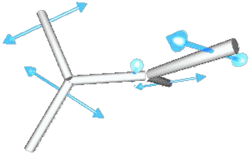
- Properties are local, relative
- Properties propagate along the body
- Control elements (neurons, sensors) are near elements under control (muscles, sticks)
- Recursive body (tree)
- Any topology of NN
- Human-friendly



XXX(XX,X)

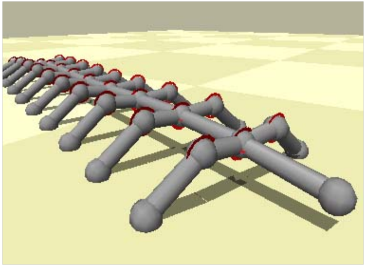


X(X,RR**X(X,X))**

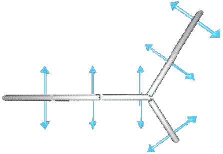


Genetics – f1 representation

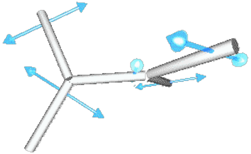
- Properties are local, relative
- Properties propagate along the body
- Control elements (neurons, sensors) are near elements under control (muscles, sticks)
- Recursive body (tree)
- Any topology of NN
- Human-friendly



XXX(XX,X)

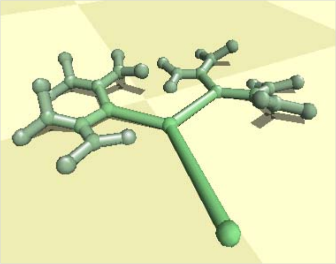


X(X,RR**X(X,X))**

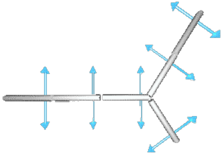


Genetics – **f1** representation

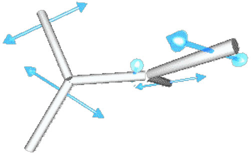
- Properties are local, relative
- Properties propagate along the body
- Control elements (neurons, sensors) are near elements under control (muscles, sticks)
- Recursive body (tree)
- Any topology of NN
- Human-friendly



XXX(XX,X)



X(X,RR**X(X,X))**



Genetics – f1 representation "modifier" genes

R r	Rotation of the branching plane by 45°
Q q	Twist of the branching plane
C c	Curvedness
L l	Length
F f	Friction
M m	Muscle strength

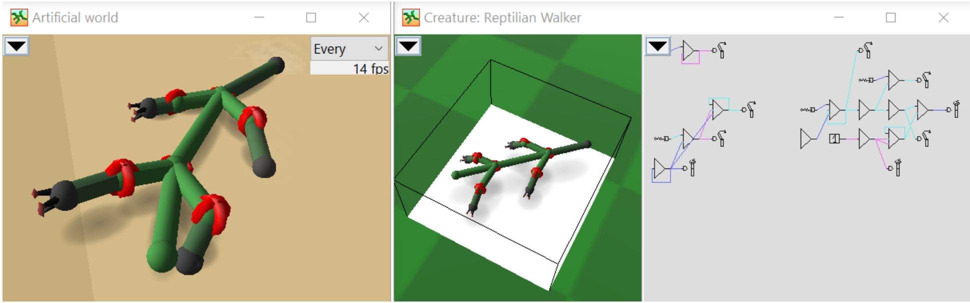
A complete description: https://www.framsticks.com/a/al_geno_f1.html

Genetics – **f1** representation example


```
db(,rrlMMMMXlFFFFCgX[|T:10.159,/:-1.442,1:3.562][@0:-51.595],FFFFlL
X[|0:2.744,-2:-3.181,-1:1.151][8:2.682],rrMMXlFFFFMMMMCgX[|T:-162.1
72,-1:8.977][@4:-0.573,3:0.724,fo:1],,,LLLXMMM(rrlMXlFFFFCgX[|T:-80.858,0:
4.784][@*:8.62],,,gX[0:657.704,-1:-3.466,-1:-346.898][|-6:2.895,fo:0.208],,,rrlMXl
FFFFCgX[N,si:999][|T:-78.873,0:2.585,-1:-2.867]))
```

Genetics – f1 representation example


```
db(,rrlMMMMXlFFFFCgX[|T:10.159,/:-1.442,1:3.562][@0:-51.595],FFFFlLX[|0:2.744,-2:-3.181,-1:1.151][8:2.682],rrMMXlFFFFMMMMCgX[|T:-162.172,-1:8.977][@4:-0.573,3:0.724,fo:1],,,LLLXMMM(rrlMXlFFFFCgX[|T:-80.858,0:4.784][@*:8.62],,,gX[0:657.704,-1:-3.466,-1:-346.898][|-6:2.895,fo:0.208],,,rrlMXlFFFFCgX[N,si:999][|T:-78.873,0:2.585,-1:-2.867]))
```

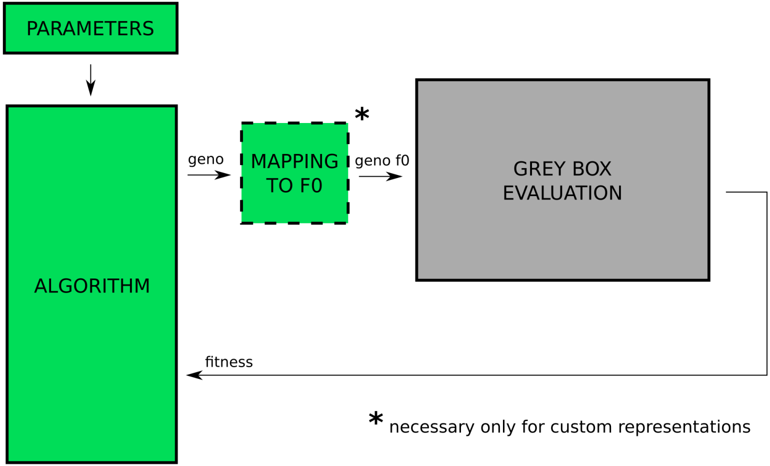


Competition

Competition goal

The competition concerns the **development of an efficient algorithm to optimize active 3D designs** (i.e., simulated agents or robots).

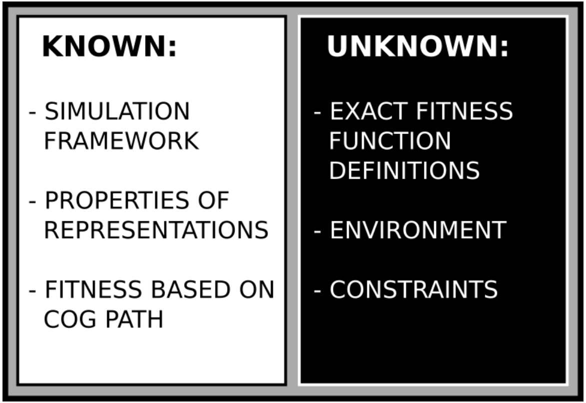
Competition



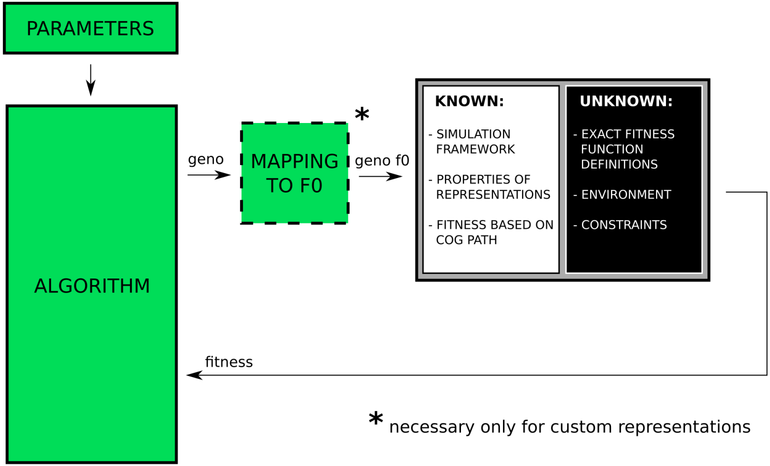
Competition

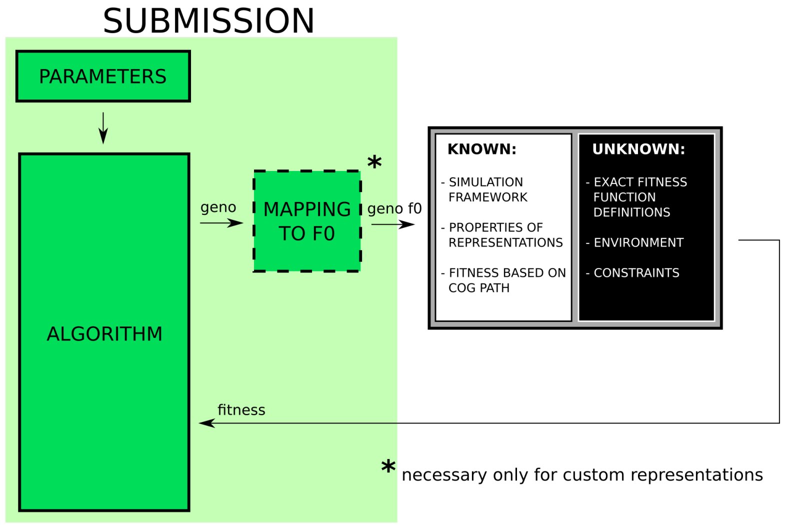
GREY BOX
EVALUATION

Competition



Competition




SUBMISSION
PARAMETERS
ALGORITHM
geno
MAPPING TO F0
*
geno f0
KNOWN:
- SIMULATION FRAMEWORK
- PROPERTIES OF REPRESENTATIONS
- FITNESS BASED ON COG PATH
UNKNOWN:
- EXACT FITNESS FUNCTION DEFINITIONS
- ENVIRONMENT
- CONSTRAINTS
fitness
* necessary only for custom representations

Fitness function examples

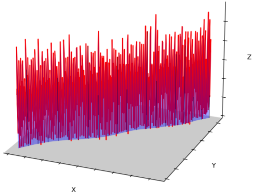
$$\textit{genotype}$$
$$\downarrow \ (\textit{simulation})$$
$$\textit{COG (center of gravity) path} = [[x_1, y_1, z_1],\ [x_2, y_2, z_2],\ \ldots,\ [x_n, y_n, z_n]]$$
$$\downarrow \ (\textit{fitness function})$$
$$\textit{fitness value}$$

Fitness function examples

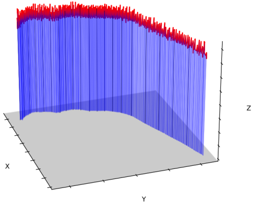
$$\textit{genotype}$$
$$\downarrow\ (\textit{simulation})$$
$$COG\ (\textit{center of gravity})\ \textit{path} = [[x_1, y_1, z_1],\ [x_2, y_2, z_2],\ \ldots,\ [x_n, y_n, z_n]]$$
$$\downarrow\ (\textit{fitness function})$$
$$\textit{fitness value}$$

Examples:

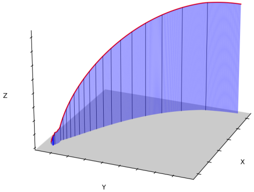
Distance



Tall runners



Rise-up



Fitness function: example formulations

from `FramsticksLibCompetition.py`


```
import numpy as np
path = np.array(path)  # COG path


if self.TEST_FUNCTION == 3:
    return np.linalg.norm(path[0] - path[-1])  # simple example:
    returns distance between COG locations of birth and death.

elif self.TEST_FUNCTION == 4:
    return np.linalg.norm(path[0] - path[-1]) * np.mean(np.maximum(0,
    path[:, 2]))  # simple example: run far and have COG high above
    ground!
```

Technicalities

- Source code in Python
- `FramsticksLib.py` – a Python class providing basic operations like mutation, crossover, and evaluation of genotypes
- `FramsticksLibCompetition.py` – same interface, but recording the highest achieved fitness and limiting the number of evaluation calls. This class is actually used when evaluating algorithm performance – participants should use it
- Public modules, libraries, and frameworks can be used
- 2 GB memory limit, single-process, single-threaded, no GPU
- Runs are terminated after 100 000 evaluations, or 1 hour of computation (excluding the time of evaluating solutions)

Judging

- 10 optimization tasks
- 30 repeated runs per task, per entry, each run returns best fitness
- These 30 best fitness values are averaged
- The resulting average is normalized taking into account other submissions
- The average of 10 normalized values constitutes the final score of the algorithm
- Winning entries must beat the baseline (a simple EA with niching)

Participants

Submissions

Three submissions:

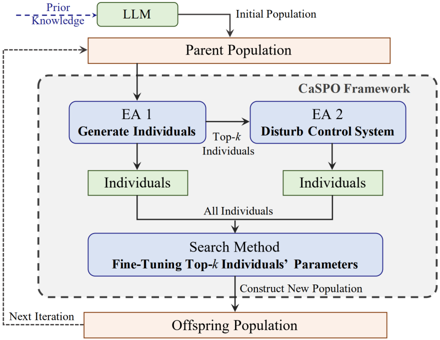
- TryBestEA
- CaSPO ("Cascaded Structure and Parameter Optimization Based on Prior Knowledge")
- AdaptMut+Diversity

Submission: TryBestEA

- This submission uses the **f1** encoding, but other encodings can be used as well
- Four different evolutionary algorithms using DEAP:
  - `eaSimple`
  - `eaMuPlusLambda`
  - `eaMuCommaLambda`
  - Custom strategy
    - Adjusting probabilities of mutation and crossover based on diversity and relative position of average fitness to median
- Perform runs using each of them (equal number of evaluations per each algorithm)
- Final result is the best result found by any of the algorithms

Submission: CaSPO

- This submission uses the **f1** encoding
- Initial population of diverse structures is generated with LLMs
- Each generation consists of three steps:
  1. Generate new individuals (EA1)
  2. Disturb control system of top-k individuals from EA1 (EA2)
  3. Fine-tune top-k individuals from combined EA1 and EA2



Details published in:
Xiang Shu, Yiyi Zhu, Renji Zhang, Xiang Xia, Bingdong Li, Hong Qian.
Automated Design Competition Technical Report: Cascaded Structure and Parameter Optimization Based on Prior Knowledge.
GECCO '24 Companion, https://doi.org/10.1145/3638530.3664054

Submission: AdaptMut+Diversity

This submission uses the **f0** encoding, but other encodings can be used as well.

Two mechanisms introduced aimed at promoting explorative capabilities:

1. Adaptive mutation strength
   - The mutation strength (i.e., the number of mutation operations applied to a genotype) is adjusted during evolution
   - Starts from mutation strength = 1.0. If the maximal fitness of the population has not changed by more than 1% for the last 4 generations, the mutation strength is multiplied by 1.1. Otherwise, it is multiplied by 0.9
   - Mutation strength is limited to the range [1, 5], and turned into an integer number of mutation operations using stochastic rounding
   - The motivation was to help the algorithm escape local optima
2. Introducing random individuals
   - Each mutation operation has a small probability (1%) of introducing a randomly generated individual to the population instead of mutating the current one
   - Allows to explore the search space more effectively – by introducing new genetic material

Population size = 50, tournament size = 5, $p_{mutation} = 0.8$, $p_{crossover} = 0.2$.

Results

Best solutions

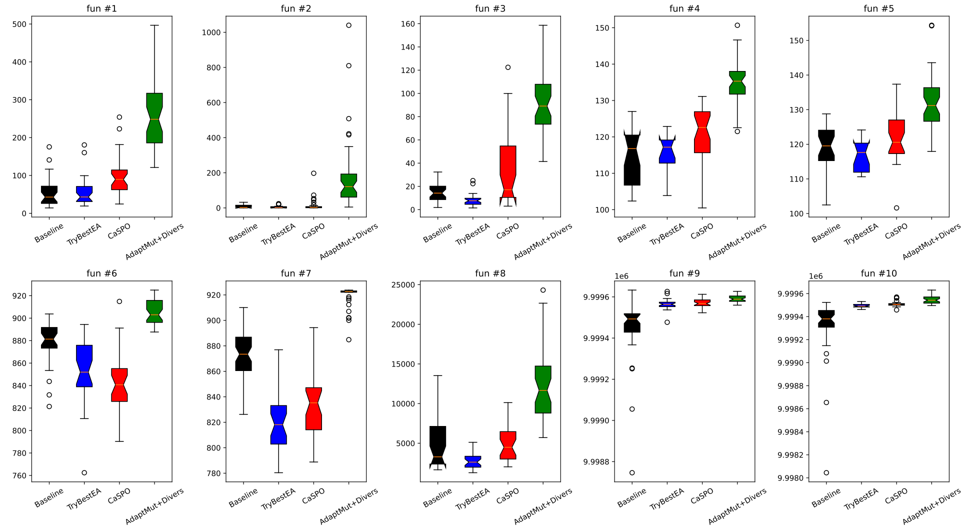
Distance:

https://www.framsticks.com/files/varia/automated-design-competition-2024-best-distance.mp4

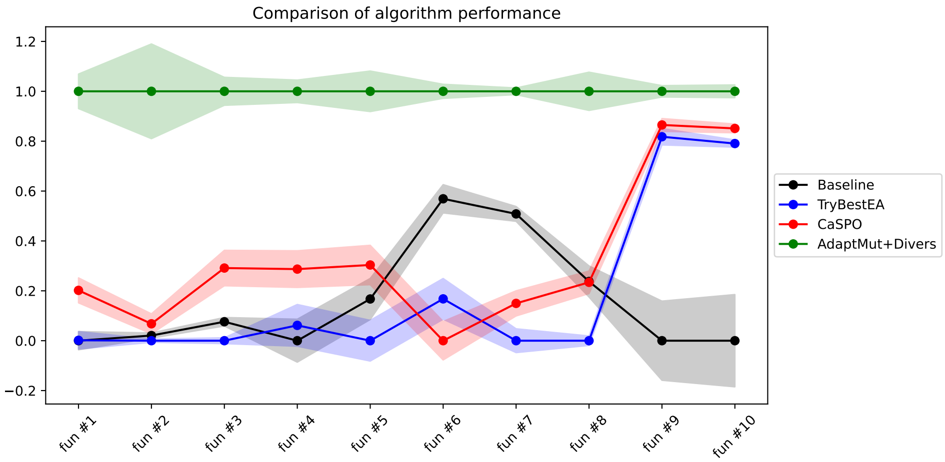
Tall runners:

https://www.framsticks.com/files/varia/automated-design-competition-2024-best-tall-runners.mp4

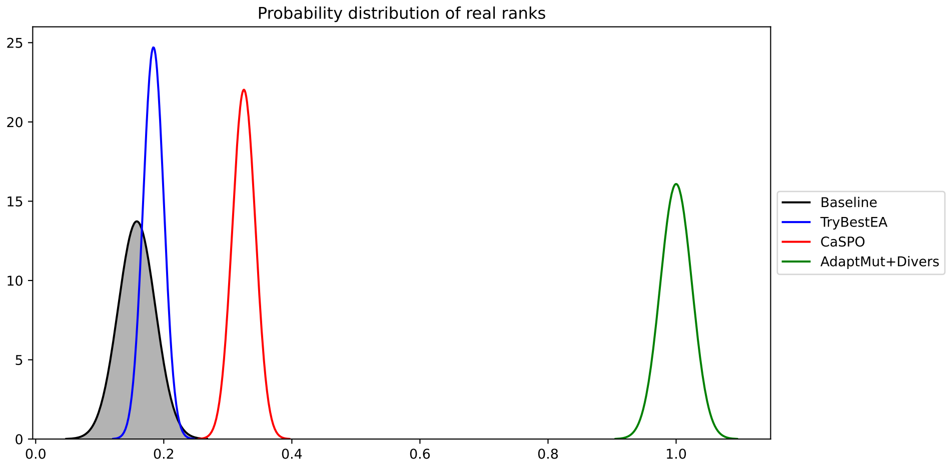
Individual benchmark tasks



Averaged normalized performance



Aggregated performance



Best algorithm: winning because of a better genetic encoding?

The best algorithm, as the only one, used the genetic representation **f0**.

Is this why this algorithm was winning?

Let us see how it performs when used with genetic representation **f1** (the one employed by all other participants).

Aggregated performance

Including the winner that uses the less performant encoding