Karlsruher Institut für Technologie

12th HPC Café

02.12.2025

Agenda HPC Café – 02.12.2025

Topics for today:

- **Xega and the Quest for Parallelization of Extended Evolutionary and Genetic Algorithms - Andreas Geyer-Schulz**
- **Updates from the HPC operations side**
- **Upcoming HPC Café**
- Discussion and open floor

As usual this format is meant to be interactive.
Don’t hesitate to ask questions!

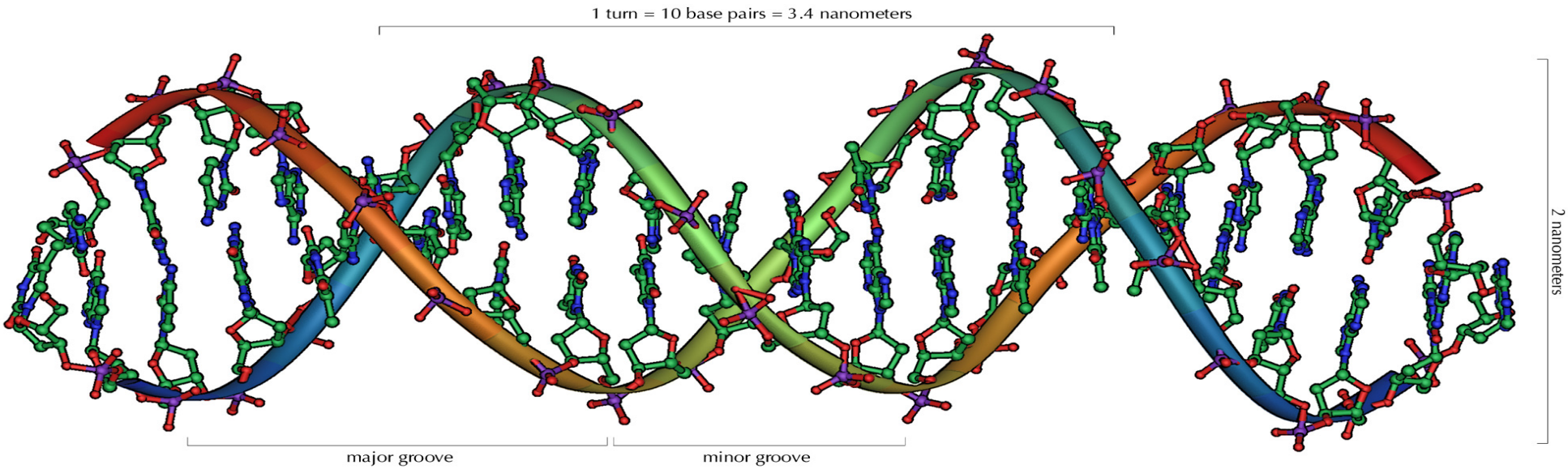
Karlsruher Institut für Technologie

Xega and the Quest for Parallelization of Extended Evolutionary and Genetic Algorithms

xega and the Quest for Parallelization of Extended Evolutionary and Genetic Algorithms

Andreas Geyer-Schulz | December 1, 2025

INFORMATION SERVICES AND ELECTRONIC MARKETS ANDREAS.GEYER-SCHULZ@KIT.EDU



Content

The Vision of Genetics

... compute with a billion (10^9) genes.

HPC-Cafe, December, 2nd, 2025, KIT, Karlsruhe)

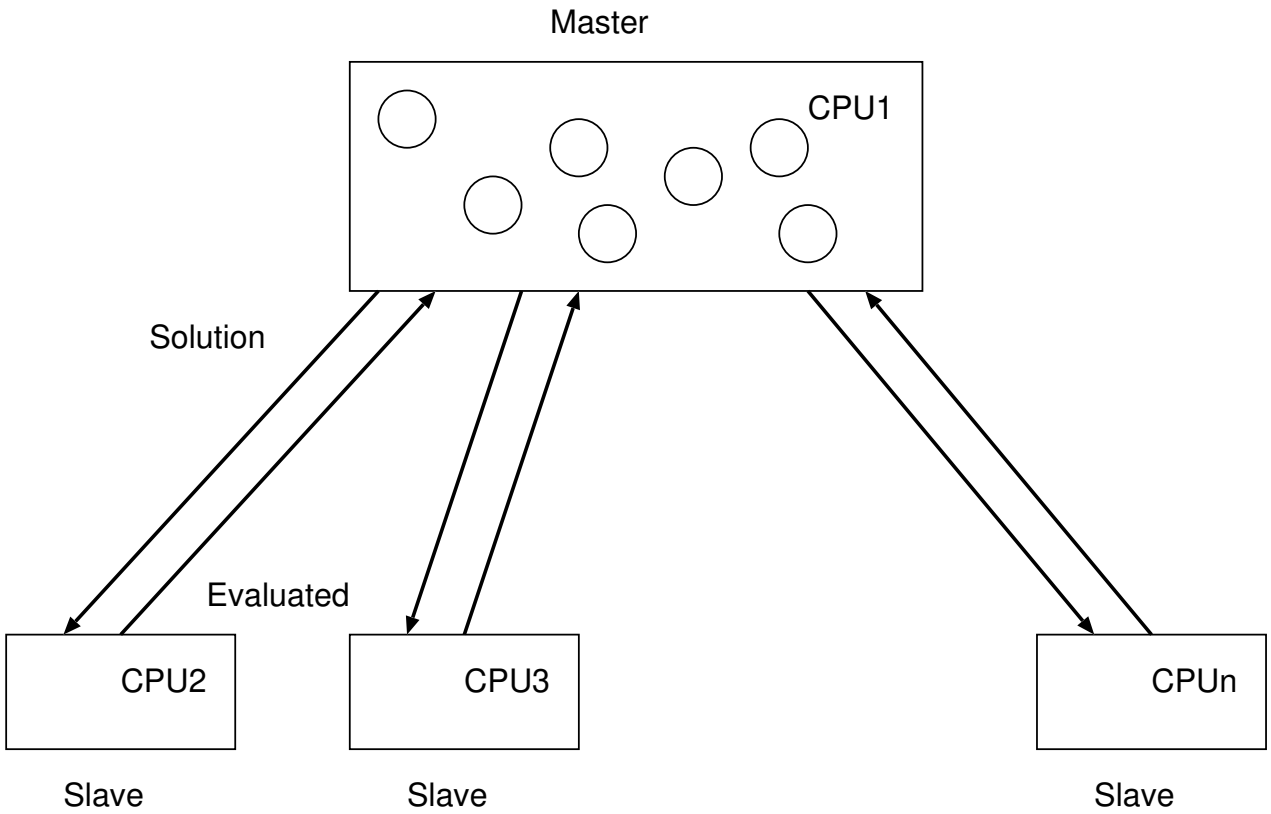
Acknowledgements. The author acknowledges support by the state of Baden-Württemberg through bwHPC.

The Phases of an Algorithm of xega I

```
Function xegaRun(a parameter list):
    InitPopulation; evalPopulation
    repeat
        nextPopulation:  foreach Gene in Population do
            ReplicateGene
            begin
                //* Abstract Operator Pipeline:  *//
                selectGene; crossGene; mutateGene; acceptGene
            end
        end
        evalPopulation:  foreach Gene in Population do
            evalGene
        end
    until Termination Condition == TRUE
```

3 Parallel Models for GAs and EAs

Global Parallelization



(as well as Parallel Island and Parallel Cellular Models)

Global Parallelization

Dominant Model for GAs and EAs, because:

- Ready-made master-slave implementations across middleware solutions are available.
- *easy* to implement with clear division of labor:
  In each generation 2 steps:
  1. Genetic machinery on master (nextPopulation): Sequential.
  2. Fitness evaluation on slaves (evalPopulation): Parallel.

Assumptions:

1. (A1) Execution cost of genetic machinery (nextPopulation) is low compared to cost of fitness evaluation (evalPopulation).
2. (A2) Available middleware works.

A single loop must be parallelized.

The R-Package xega: e(x)tended (e)volutionary and (g)enetic (a)lgorithms

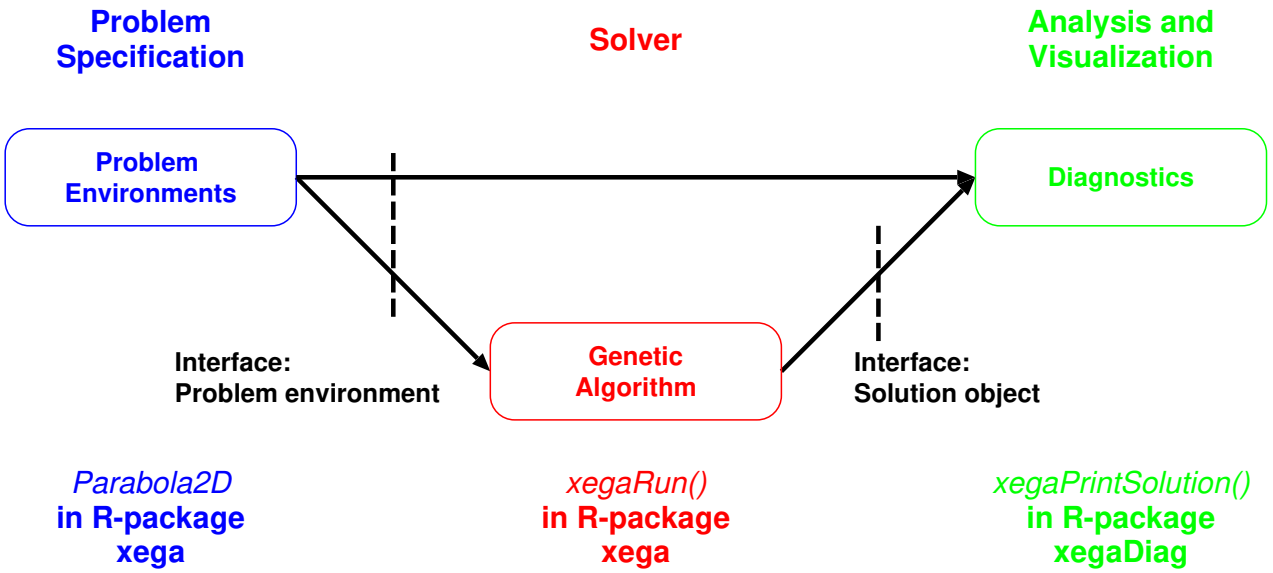
- Provides 64 million structurally different heuristic algorithms (Families: Evolutionary algorithms, genetic algorithms, differential evolution, simulated annealing, grammar-based genetic programming, grammatical evolution, grammatical differential evolution, permutation-based genetic algorithms.)
- Implemented in R-script (a kind of 2nd generation LISP).
- xega (0.9.0.18) [Geyer-Schulz, 2024]:
  `https://CRAN.R-project.org/package=xega`

```
install.packages("xega")
```

- Developer's version: `https://github.com/ageyerschulz/xega`
- xega's architecture: [Geyer-Schulz, 2025]

Step 1: Installation.

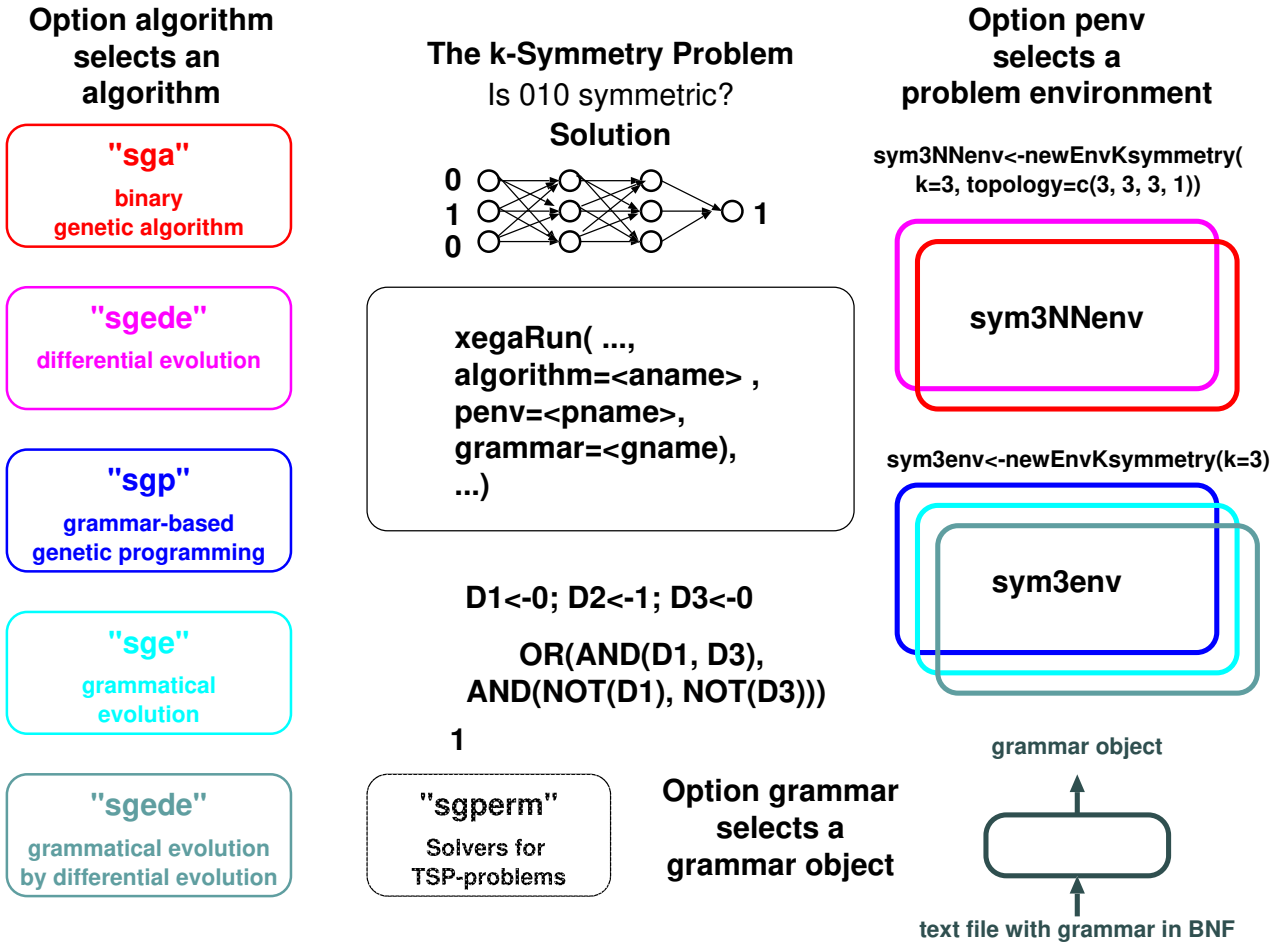
The xega Analysis Pipeline



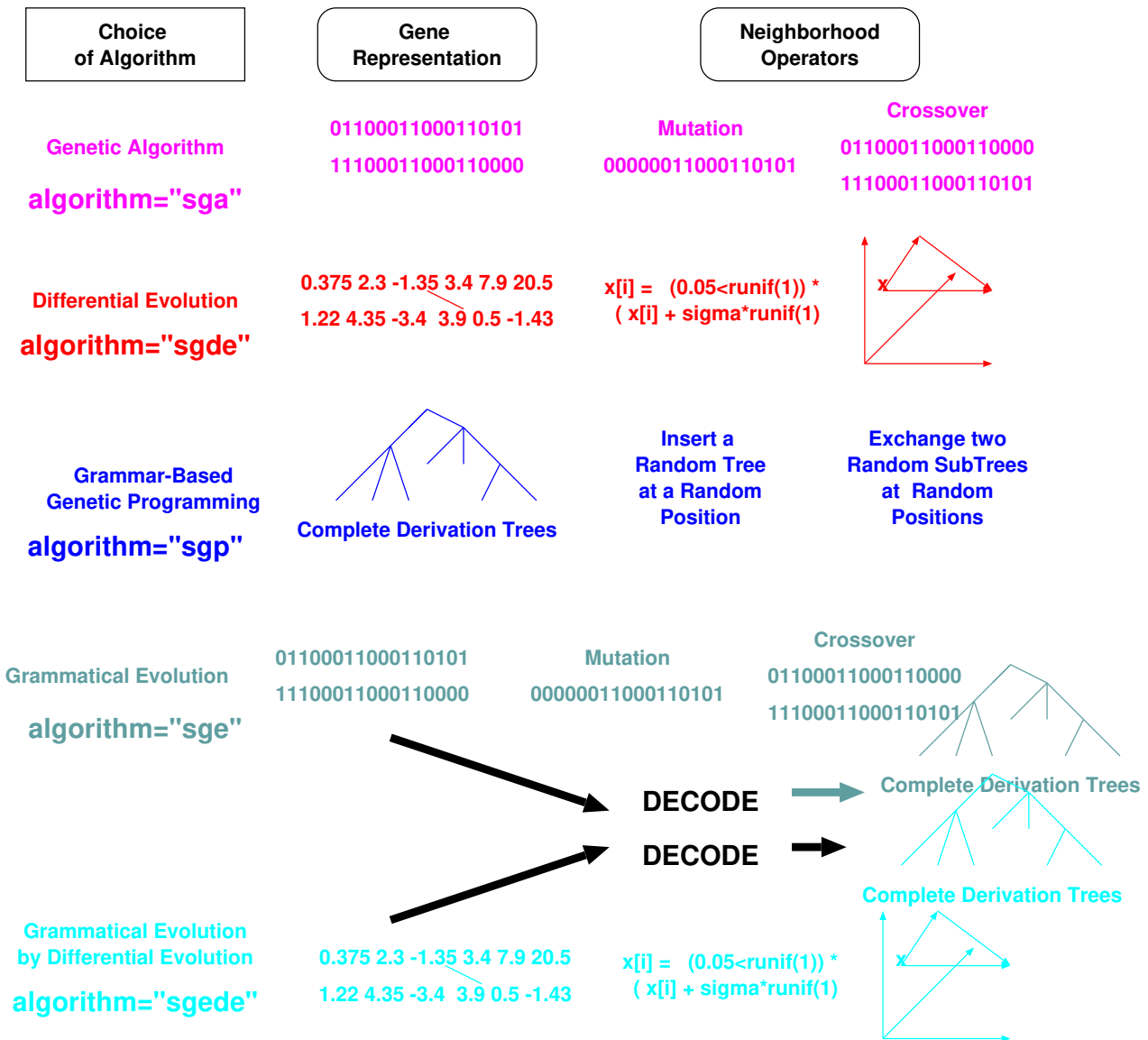

```
> library(xega); library(xegaDiag)
Loading required package: parallelly
> a<-xegaRun(penv=Parabola2D, algorithm="sga", max=FALSE, verbose=0)
> xegaPrintSolution(a)
Min   Parabola2D !
Fitness: -0.0008741791 f(Parameters): -0.0008741791
Parameters: 0.02612259 -0.01384879
Time used: 0.1407065 seconds.
```

An End User Session: Find the minimum of a 2-D Parabola!

xega Algorithms and Problem Environments



xega Algorithms



xega's Approach to Parallelization (I)

Redefine the lapply statement in `xegaPopulation::xegaEvalPopulation()`:

```
pop<-lF$lapply(pop, lF$EvalGene, lF=lF)
```

`xega` provides:

1. a set of preconfigured redefinitions of the lapply statement (multicore, future, cluster) with synchronous and asynchronous communication. E.g. for multicore with synchronous communication:

```
MClapply<-function(pop, EvalGene, lF)
{    parallel::mclapply(pop, EvalGene, lF=lF,
                        mc.cores=max(1, lF$Cores()), mc.set.seed = TRUE)
}
```

2. the possibility of injecting user-defined parallel lapply statements (e.g. for using mpi).

The Problem: Assumption A1 Does Not Hold ...

Problem TSP a280 (Multicore)	Run 1-a	Run 2	Run 3
Population	320	500	700
Generations	100	200	300
NextPopulation - Sequential (s)	1503.87	4677.60	11576.55
xega Main Loop (s)	1504.93	4903.27	11588.47
Time Spent in NextPopulation (%)	99.93	95.40	99.90

Table 1: Percentage of execution time in sequential code for the symmetric TSP problem a280 (from the TSPLIB).

Reason (for this example):

- TSP heuristics which evaluate the fitness of a gene as genetic operators.

Implication: Gains from parallelization are **very, very small** for most algorithms in xega.

... for Almost All Algorithms in xega

	Experiments (AMD)		
	Main (s)	Replication (s)	% Seq
Grammatical Evolution (GE)	7700.09	25.64	0.33
Genetic Algorithm (GA)	1370.50	26.28	1.92
Grammar-Based Genetic Programming (GP)	8952.49	1167.99	13.05
Differential Evolution (DE)	1455.06	1445.90	99.37
Grammatical Differential Evolution (GDE)	7862.46	7832.93	99.62
GA with Permutation Operators (GAperm)	8554.43	8543.16	99.87

Table 2: Percentage of Execution Time Spent in Sequential Code. Ranked and Grouped by % Seq(uential).

.

High Potential for Parallelization with Pipeline Compilation!

Pipeline Compilation

Definition

Pipeline compilation is the process of transforming an abstract genetic operator pipeline into a function closure.

Components of an R function:

1. An argument list.
2. The function body (a parsed R statement).
3. The environment of the function is the environment that was active at the time that the function was created. Any symbols bound in that environment are captured and available to the function.

From R Code ...

Input: *population*: A vector of genes.
fitness: A vector of reals.
Output: *gene*: A gene.

Function `ReplicateGene(`*population, fitness*`)`**:**
  gene ⟵ `selectGene(`*population*, *fitness*`)`;
  if `rand()` $<$ *crossrate* **then**
    gene2 ⟵ `selectGene(`*population*, *fitness*`)`;
    gene ⟵ `crossGene(`*gene*, *gene2*`)`
  end
  if `rand()` $<$ *mutrate* **then**
    gene ⟵ `mutateGene(`*gene*`)`
  end
  return *gene*

... to Creating Function Closures

Input: *population*: Vector of genes.
fitness: Vector of reals.
lF: Local function list
Output: *gene*: A gene.

Function `ReplicateGenePipeline(`*population, fitness, lF*`):`

```
    gene ⟵ lF$selectGene (population, fitness, lF);
    cross ⟵ rand() < lF$crossrate();
    mut ⟵ rand() < lF$mutrate();
    if cross and not(mut) then
        gene1 ⟵ lF$selectMate (population, fitness, lF);
        return newCrossPipeline (gene, gene1, lF)
    end
    if not(cross) and not(mut) then
        return newPipeline(gene, lF)
    end
    if not(cross) and mut then
        return newMutPipeline(gene, lF)
    end
    if cross and mut then
        gene1 ⟵ lF$selectMate (population, fitness, lF);
        return newCrossMutPipeline(gene, gene1, lF)
    end
```

The Constructor for a Function Closure with Crossover and Mutation I

```
newCrossMutPipeline<-function(g, g1, lF)
{ Pipeline<-function(lF)
{  OPpip<-function(g, lF)
    { g2<-lF$CrossGene(g, g1, lF)[[1]]
      lF$MutateGene(g2, lF)}
   lF$EvalGene(lF$Accept(OPpip, g, lF), lF)}
  rlang::env_unbind(environment(Pipeline), c("lF"))
return(Pipeline) }


a<-newCrossMutPipeline(g, g1, lFxegaGaGene)
```

produces the closure

```
print(a)
function (lF)
{
    OPpip <- function(g, lF) {
        g2 <- lF$CrossGene(g, g1, lF)[[1]]
        lF$MutateGene(g2, lF)
    }
    lF$EvalGene(lF$Accept(OPpip, g, lF), lF)
}
<bytecode: 0x557ed58fbb48>
<environment: 0x557ed58fe0f8>
```

The Constructor for a Function Closure with Crossover and Mutation II

with the environment

```
>as.list(environment(a))
$Pipeline
function (lF)
{    OPpip <- function(g, lF) {
         g2 <- lF$CrossGene(g, g1, lF)[[1]]
         lF$MutateGene(g2, lF) }
     lF$EvalGene(lF$Accept(OPpip, g, lF), lF)
}
<bytecode: 0x557ed58fbb48>
<environment: 0x557ed58fe0f8>

$g
$g$evaluated
[1] FALSE

$g$evalFail
[1] FALSE

$g$fit
[1] 0

$g$gene1
 [1] 0 1 1 0 1 0 1 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 1 1 0 1 0 1 1 0 0 1 1 0 1
```

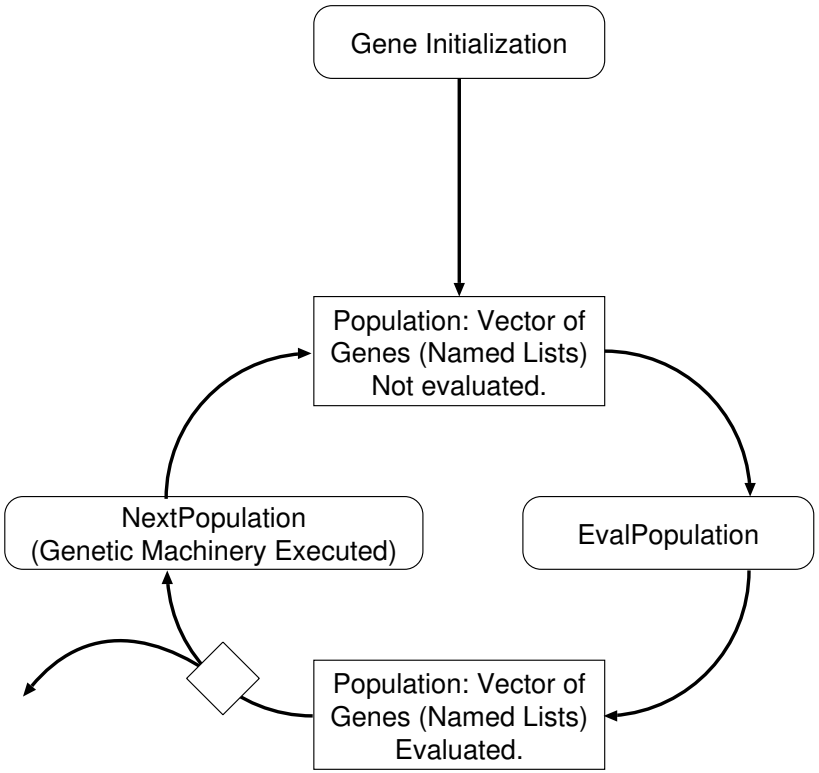
The Constructor for a Function Closure with Crossover and Mutation III

```
$g1
$g1$evaluated
[1] FALSE

$g1$evalFail
[1] FALSE

$g1$fit
[1] 0

$g1$gene1
 [1] 0 1 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 0
```

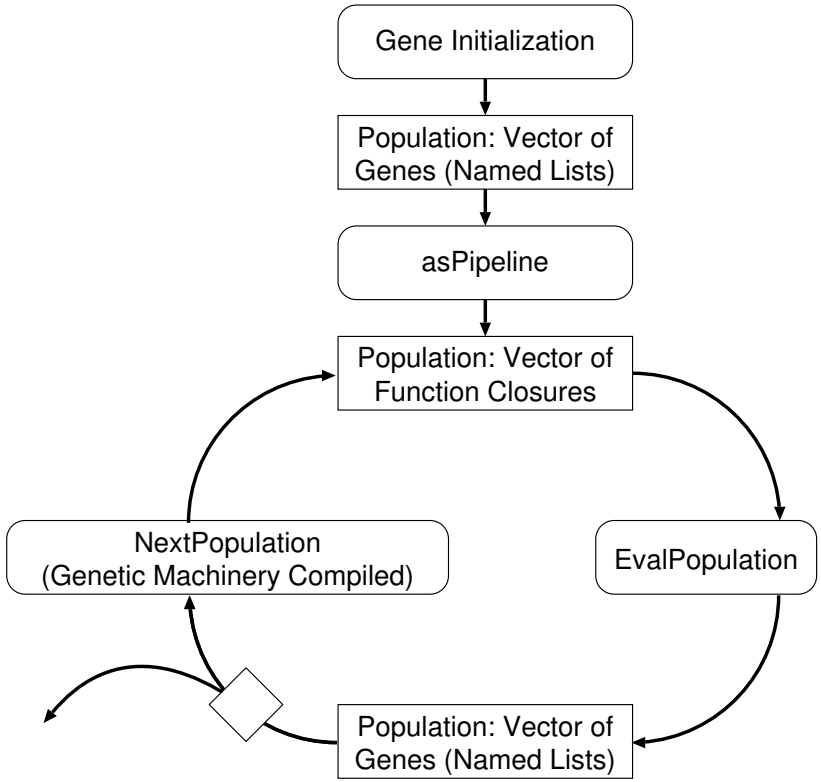
For additional (full) examples:

```
library(xegaGaGene)
example(newPipeline)
example(newMutPipeline)
example(newCrossPipeline)
example(newCrossMutPipeline)
```

The Gene Representation Cycle



Operator Pipeline Executed



Operator Pipeline Compiled

The Phases of an Algorithm of xega (Pipeline Compilation)

```
Function xegaRun(a parameter list):
    InitPopulation; asPipeline; evalPopulation
    repeat
        nextPopulation:  foreach Gene in Population do
            ReplicateGene
            begin
                Compile Abstract Operator Pipeline
            end
        end
        evalPopulation:  foreach Closure in Population do
            evalClosure
            begin
                selectGene; crossGene; mutateGene; acceptGene;
                evalGene
            end
        end
    until Termination Condition == TRUE
```

The Shift in Execution Time ...

Phases of xega	A: No Pipeline (s)	B: Pipeline (s)
tMainLoop	325.88	318.91
tInitPopulation	0.01	0.01
tNextPopulation	325.10	0.28
tEvalPopulation	0.74	318.58
tObservePopulation	0.01	0.01
tSummaryPopulation	0.01	0.01

Table 3: Profile of Run 1-b for TSP problem a280. Rounded to 2 Decimals. Execution model: **Sequential.** Population: 320. Generations: 100. Crossover Rate: 0.1. Mutation Rate: 0.1. Experiment (INTEL)

... and Gains from Parallelization

Phases of xega	A: No Pipeline (s)	B: Pipeline (s)	A/B	B/A
tMainLoop	30215.81	5069.88	**5.96**	0.17
tInitPopulation	0.02	0.01		
tNextPopulation	30193.24	10.79		
tEvalPopulation	21.96	5058.00		
tObservePopulation	0.25	0.18		
tSummaryPopulation	0.04	0.03		

Table 4: Profiles of Run 5 for TSP problem a280. Rounded to 2 Decimals. Execution model: Multicore. Population: 1200. Generations: 500. Crossover Rate: 0.1. Mutation Rate: 0.1. Experiment (INTEL) with 20 virtual cores

Experimental Setup

Treatment	Execution Model	Pipeline Compilation
(1)	Sequential	No
(2)	Sequential	Yes
(3)	MultiCore	No
(4)	MultiCore	Yes

Table 5: Experimental Treatments. Population size: 2400. Generations: 200.

For each algorithm:

Algorithm	Problem	Solution	Main Operations
GE	8-Symmetry	Boolean Formula	Logic, integer
GA	8-Symmetry	Neural Network	Floating point
GP	8-Symmetry	Boolean Formula	List, logic, integer
DE	8-Symmetry	Neural Network	Floating point
GDE	8-Symmetry	Boolean Formula	Logic, floating point
GAperm	TSP lin105	Permutation	Integer, floating point

Genetic Programming

Tr.	Alg.	Exec.	Pipe	$\overline{Speedup}$	$\overline{Main}$ (s)	$\overline{Next}$ (s)	$\overline{Eval}$ (s)	x
(1)	GP	Seq	No	28.00	7063.52 (σ: 62.01)	943.73 (σ: 8.96)	6119.02 (σ: 55.81)	15.00
(2)	GP	Seq	Yes	28.04	7073.27 (σ: 73.98)	11.22 (σ: 0.25)	7061.28 (σ: 73.92)	15.00
(3)	GP	MC	No	4.27	1078.34 (σ: 15.84)	906.79 (σ: 15.19)	170.66 (σ: 3.19)	15.00
(4)	GP	MC	Yes	1.00	252.29 (σ: 2.42)	19.55 (σ: 0.65)	231.67 (σ: 2.55)	15.00
Experiment HPC with 40 virtual cores								

Table 6: 8-Symmetry Problem. GP: Grammar-Based Genetic Programming. Optimal Solution: 0. 50 Trials.

Differential Evolution

Tr.	Alg.	Exec.	Pipe	$\overline{Speedup}$	$\overline{Main}$ (s)	$\overline{Next}$ (s)	$\overline{Eval}$ (s)	x
(1)	DE	Seq	No	8.05	1165.35 (σ: 14.42)	1158.68 (σ: 14.36)	6.48 (σ: 0.07)	11.43
(2)	DE	Seq	Yes	8.07	1167.99 (σ: 18.86)	15.65 (σ: 0.23)	1152.14 (σ: 18.74)	11.43
(3)	DE	MC	No	8.36	1210.26 (σ: 16.73)	1194.68 (σ: 16.74)	15.35 (σ: 0.22)	11.43
(4)	DE	MC	Yes	1.00	144.76 (σ: 1.52)	20.10 (σ: 0.28)	124.31 (σ: 1.44)	14.01
Experiment HPC with 40 virtual cores								

Table 7: 8-Symmetry Problem NN. DE: Differential Evolution. Optimal Solution: 0. 50 Trials.

Summary of Experiments

	Sequential		Parallel	
Pipeline:	FALSE	TRUE	FALSE	TRUE
Algorithm	Speedup (1)	Speedup (2)	Speedup (3)	Speedup (4)
GE	29.85	27.98	0.95	1.00
GA (NN, FP)	8.24	8.24	1.04	1.00
GP	28.00	28.04	4.27	1.00
DE (NN, FP)	8.05	8.07	8.36	1.00
GDE	22.85	22.81	23.83	1.00
GAperm (TSP, FP)	9.17	9.22	9.41	1.00

Pipeline compilation improves the performance of 5 out of 6 algorithm families.

Limits of MultiCore (SMP) Parallelization for R

- Maximum of 125 cores. (TCP/IP 7bit address in sendqueue).
- With out-of-the-box parallelization,
  not possible to use **all cores** on one node.
- Floating point support of HPC nodes too limited.

With 1000 genes per core,
10^4 cores to go ...

Assumption 2 does not hold:

- **R interface problems:** Rmpi does not serialize function closures.
  Consequence: Pipeline compilation does not work.
  Remedy: base::serialize() and base::unserialize() work with function closures.
  Modify apply function!
- Fix leads to **new problems**:
  - **Out of memory**:
    - Fix: “Manual garbage collection” leads to performance degradation.
    - Potential fix: Allocate different memory size to master and slave processes.
      How do I specify this in Slurm script?

Rmpi and mpi II

- **OpenMPI UCX 1.18 transport layer bugs** in scatter and gather operations **for large and complex R objects** (e.g. complete derivation trees).
  - Use older OpenMPI version without bugs. (Which? Who installs?)
  - Rewrite distribution code: Communicate only filenames via rmpi. Rest of the communication via base::saveRDS() and base::readRDS() over the file system.
  - Think about the architecture and minimize the need to communicate.

Unsolved problems!

Paper and Code

- Geyer-Schulz, A. and Zamani Shandiz, M. (2025) Compiling Abstract Genetic Operator Pipelines in the R Package xega. [Geyer-Schulz and Zamani Shandiz, 2025]
- Code: `https://github.com/ageyerschulz/xegaPipelineExperiments`
- xega 0.9.0.18 `https://CRAN.R-project.org/package=xega`

References I


Geyer-Schulz, A. (2024).
xega: Extended evolutionary and genetic algorithms.
Technical Report 2024-10, IISM, KIT, Karlruhe.


Geyer-Schulz, A. (2025).
xegaX. A family of R-packages for genetic and evolutionary algorithms with multiple genome representations.
Archives of Data Science, Series A, 10(1):1 – 37.


Geyer-Schulz, A. and Zamani Shandiz, M. (2025).
Compiling abstract genetic operator pipelines in the R-package xega.
Evolutionary Computation.
Submitted.

Karlsruher Institut für Technologie

Updates from the HPC operations side

Maintenance and Transition to HoreKa 2

- Maintenance work in the HPC building at Campus North
  - Extensive electrical work in preparation for HoreKa 2
  - Update: **No downtime in 2025** of the systems
  - Current schedule: **Downtime in CW 7 in 2026**
    - Starting Monday, 9 February 2026
    - Estimated duration one week

- Transition to HoreKa 2
  - Most importantly: you will **be able to use your granted compute time**
  - There will **always be CPU and accelerated resources available**
  - Transition to the new system will happen at a specific date (tbd)

  → We will keep you informed about any steps you might need to do in advance

Karlsruher Institut für Technologie

Upcoming HPC Cafés

Next HPC Café

- Next HPC Café planed for beginning of February
- Starting March 2026 new timeslot for the HPC Café
  - Every **first Thursday of the month 10:00 am**

- As always, open to hear ideas for topics and talks, from …
  - Yourself
  - Colleagues and collaborators
  - Also just expressions for “topics of interest”

The NHR@KIT, bwHPC-S5 @KIT and SCS team wished you a merry Christmas time and a happy new year!