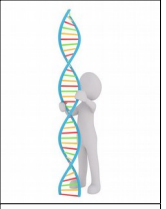
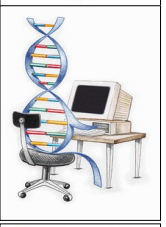
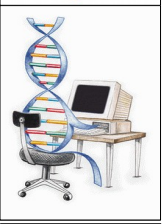







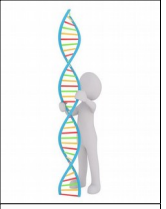
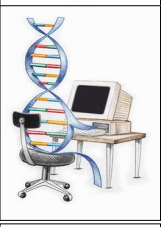

EvoHyp – A Java Toolkit for Evolutionary Algorithm Hyper-Heuristics

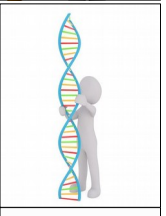
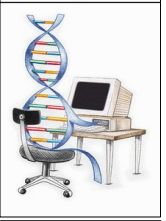
Evolutionary Algorithm Hyper-Heuristics

- Selection hyper-heuristics
  - genetic algorithms
  - explores the space of low-level combinations
- Generation hyper-heuristics
  - Components – existing heuristics, decomposed existing heuristics, problem characteristics
  - Arithmetic operators
  - Branching/conditional operators
  - Functions vs. rules

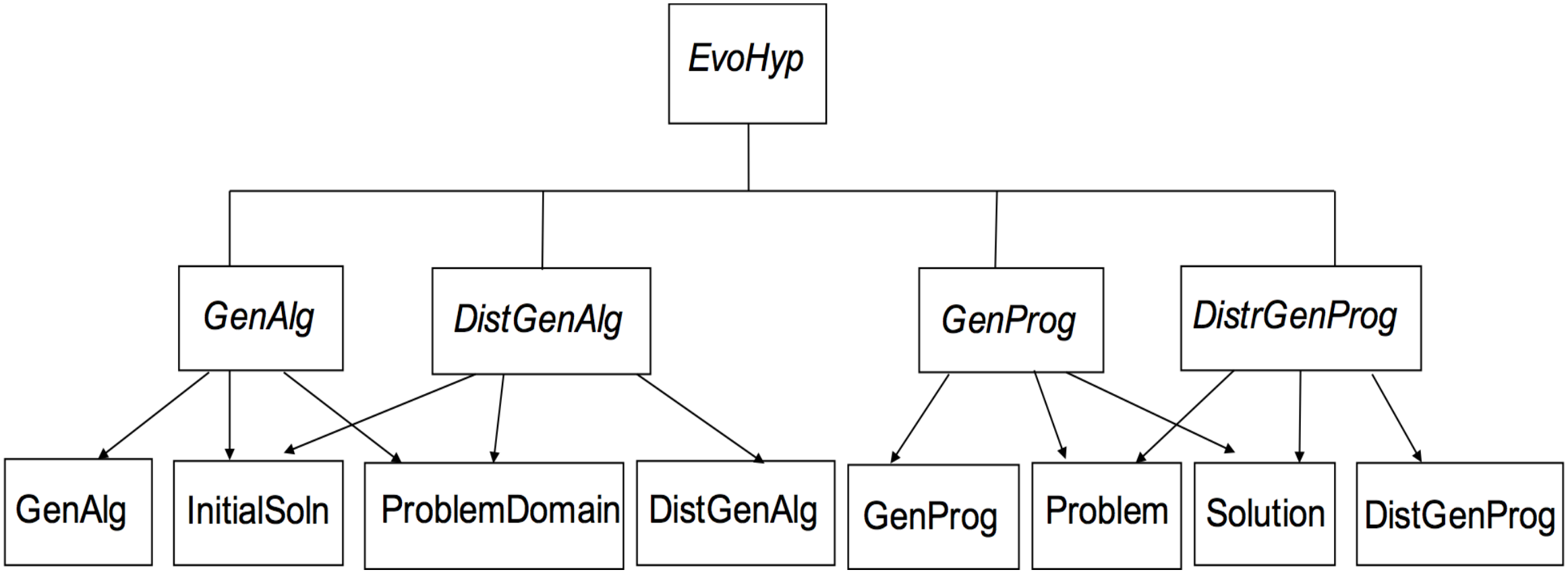
EvoHyp

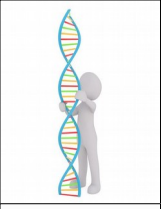
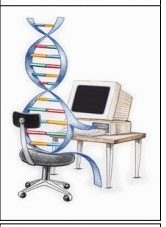
- Provides a genetic algorithm selection hyper-heuristic (*GenAlg*).
- Provides a genetic algorithm generation constructive hyper-heuristics (*GenProg*).
- Provides distributed versions of *GenAlg* and *GenProg.*
- The user has to implement the problem domain.
- How does *EvoHyp* differ from existing hyper-heuristic toolkits?

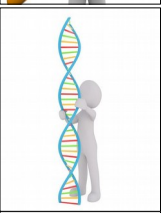
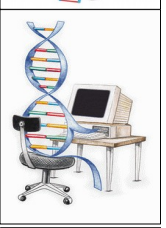



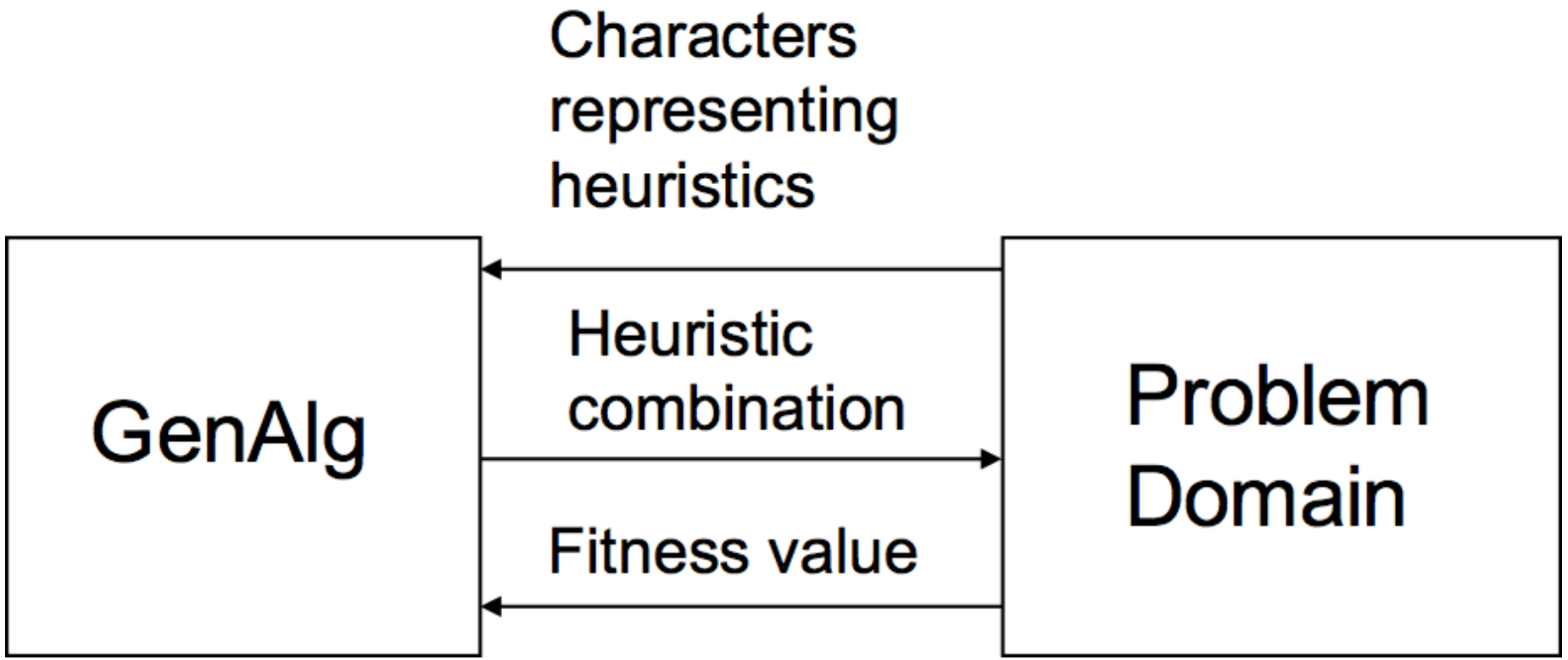

EvoHyp


EvoHyp
GenAlg
DistGenAlg
GenProg
DistrGenProg
GenAlg
InitialSoln
ProblemDomain
DistGenAlg
GenProg
Problem
Solution
DistGenProg


GenAlg Overview

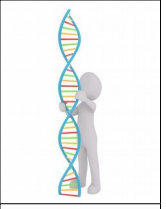
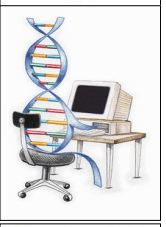


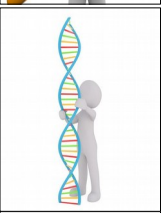
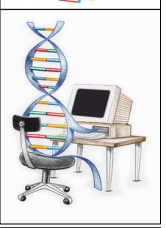
GenAlg Overview

- Caters for selection constructive and selection perturbative hyper-heuristics.
- Implements a generational genetic algorithm.
- Each chromosome is a sequence of low-level heuristics, e.g. *efab*
- Fitness of each chromosome is the “objective value” of the resulting solution.
  - user defines a `evaluate` function as part of the problem domain
- Tournament selection us used to select parents.
- Mutation and crossover are used to produce offspring.

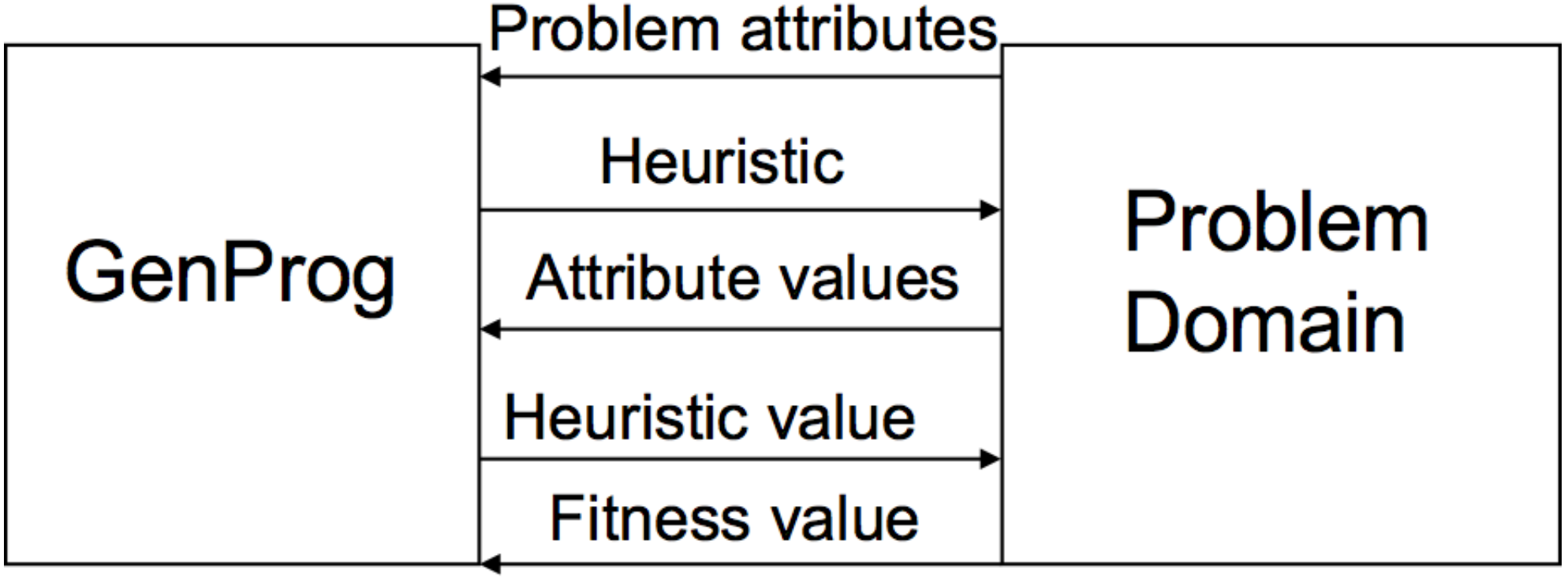
Using *GenAlg*

- Implement the *InitialSolution* class
  - Comparison of two solution to indicate the fitter
  - Fitness of the heuristic combination
  - Heuristic combination
  - Solution produced by the combination
- Implement the *ProblemDomain* class
  - Evaluation of the heuristic combination
- Use the *GenAlg* class to implement the selection hyper-heuristic


GenProg Overview



GenProg Overview

- Generational genetic programming algorithm.
- Evolves and arithmetic function or an arithmetic rule.
- Terminal set : problem attributes
- Function set : arithmetic operators, if-then-else
- Grow method used to create initial population.
- Fitness of each individual is the “objective value” of the resulting solution.
- Tournament selection is used to choose parents.
- Mutation and crossover generate offspring.

Using *GenProg*

- Implement the *Solution* class
  - Comparison of two solution to indicate the fitter
  - Fitness of the heuristic combination
  - Heuristic combination
  - Solution produced by the combination
- Implement the *Problem* class
  - Evaluation of the heuristic combination
  - Set the attributes
  - `Evaluator` class
- Use the *GenProg* class to implement the selection hyper-heuristic

DistrGenAlg and *DistrGenProg*

- Distributed versions of *GenAlg* and *GenProg*
- The algorithm is distributed over a multicore architecture.
- Initial population generation and evaluation is distributed
- Creation of offspring and evaluation is distributed.

Future Extensions

- Including options for the GA and GP, e.g. steady state
- Including other evolutionary algorithms.
- Genetic programming has primarily been used for this.
- Parameter tuning.
- Generation perturbative hyper-heuristic.
- Catering for user needs and input

EvoHyp URL

https://sites.google.com/view/evohyp/home

http://titancs.ukzn.ac.za/EvoHyp.aspx