Plug-and-Play hyper-heuristics: an extended formulation
Patricia Ryser-Welch and Julian F. Miller

Our motivations: improving the design of algorithms and understanding their behaviour.

Proposed extended formulation

We are suggesting to merge the elements of the “Algorithm Selection Problem” and the “Two-level model” together to form a more explicit architecture. The first method describes explicitly the learning mechanism taking place during the optimisation of algorithms; those elements are in dark gray in the diagram below. The second model separates the search method from a specific problem domain; the two levels in green. Plug-and-Play hyper-heuristics add a new level of indirection to extends the Domain barrier level interface into a new module, with its own full set of components.

Benefits

A high modularity
The problem domain and the algorithm optimisation are kept distinct from one another.

More independence
Each component can be substituted with another one. The new Interpreter Space is customised to the problem and the optimisation method, mapping the instruction set of the hyper level to the operations of the base level.

More transparency
All the functions used in a hyper-heuristics framework are clearly explained.

Experimental results - Evolving Evolutionary Algorithms solving OnesMax

1. Learn
At the Hyper level, a learning mechanism generates the algorithms. Each algorithm is encoded in a CGP graph. The search space of all possible algorithms is explored with Evolutionary strategies.

2. Analyse and choose
The performance of all the generated algorithms is assessed from the OnesMax solutions obtained after a short run. The structure of the algorithms is studied, to ensure each algorithm is complete.

3. Apply
The more promising algorithms are executed many times to solve OnesMax problems instances. These solutions are compared against solutions obtained with a standard evolutionary algorithms.

The Role of a new explicit component - An Interpreter

The interpreter matches the opCodes encoded in the CGP graphs against the operations defined in the base level and execute them. It is unique to each problem. Consequently, Plug-and-Play hyper-heuristics offers a higher-level modularity and a more problem-independent learning mechanism.

Operators - Mapping operations

(0,op_0) CrossoverOnePoint()
(1,op_1) CrossoverTwoPoints()
(2,op_2) CrossOverUniform()
(3,op_3) Mutate()
(4,op_4) MutateHillClimbing()
(5,op_5) MutateNonUniform()
(6,op_6) ReplaceWeakest()
(7,op_7) ReplaceRandom()

Some examples of generated hybrid algorithms

For each algorithm a population of OnesMax solutions is randomly created and evaluated. The operations are iteratively repeated to find an OnesMax solutions. Each time replaceRandom and replaceWeakest is executed on a population of offsprings, the new offsprings are evaluated. The budget of 751 evaluation is then reduced by 1 unit.

Algorithm A:
Mutate - ReplaceWeakest - CrossoverUniform - ReplaceWeakest
Algorithm B:
ReplaceWeakest - MutateHillClimbing - MutateHillClimbing - ReplaceRandom - CrossoverUniform - ReplaceRandom
Algorithm C:
MutateHillClimbing - MutateNonUniform - MutateHillClimbing - ReplaceWeakest

The algorithm applying all hillclimbing operators solves consistently the OnesMax problem with 1000 bits. A budget of 751 evaluations was given to add a competitive edge. Algorithm C solved OnesMax of approximately 300 evaluations in average, the others algorithms require a larger budget.