A Bi-population Particle Swarm Optimizer for Learning Automata based Slow Intelligent System

Mohammad H. Mofrad
University of Pittsburgh
Thursday, December 08, 2016
Preface

• Slow Intelligence System (SIS)
• Particle Swarm Optimization (PSO)
• Learning Automata (LA)
• Adaptive Intelligence Optimizer (AIO)
Slow Intelligence System (SIS)

- **Enumeration** (-enum<) of the different available solutions until finding the optimal solution

- **Propagation** (=prop+) of the achieved new information from the new solutions within a body of feasible solutions.

- **Adaptation** (+adap=) of the current solutions using the effective information gained from the elite solutions.

- **Elimination** (>elim-) of the worst solutions that exist in the problem space.

- **Concentration** (>conc=) on the elite solutions to produce new promising solutions.
Particle Swarm Optimization (PSO)

- Inspired from movement of animals
- \( V_i = w \ V_i + c_1 \ r_1 \ (pbest_i - X_i) + c_2 \ r_2 \ (gbest - X_i) \)
- \( X_i = X_i + V_i \)

Learning Automata (LA)

- Belongs to the reinforcement learning family

\[ p_j(n+1) = \begin{cases} 
  p_j(n) + a \times (1 - p_j(n)) & j = i \\
  p_j(n) \times (1 - a) & \forall j \mid j \neq i 
\end{cases} \]

\[ p_j(n+1) = \begin{cases} 
  p_j(n) \times (1 - b) & j = i \\
  b \times (r - 1) + (1 - b) \times p_j(n) & \forall j \mid j \neq i 
\end{cases} \]

Adaptive Intelligence Optimizer (AIO)

- 2 PSO populations
- SIS framework
- LA framework
SIS + PSO

• Mapping SIS’s operators to PSO formula
  • \( \text{cycle}_1: [\text{guard}_{1,2}] P_0 \rightarrow \text{enum}<P_1 = \text{prop} + P_2 > \text{elim} - P_3 > \text{conc} = P_4 \)
  • \( \text{cycle}_2: [\text{guard}_{2,1}] P_0 \rightarrow \text{enum}<P_1 = \text{prop} + P_2 + \text{adap} = P_3 > \text{elim} - P_4 > \text{conc} = P_5 \)

• Simulating SIS’s slow & quick decision cycles
  • \( w = w_{\text{max}} - (0.75i (w_{\text{max}} - w_{\text{min}})/i_{\text{max}}) \)
  • \( w = w_{\text{max}} - (i (w_{\text{max}} - w_{\text{min}})/i_{\text{max}}) \)
Implementing SIS’s TDR system using AIO

\[ T \]

\[ \begin{array}{ccc}
L_1 & L_2 & L_n \\
\downarrow & \downarrow & \downarrow \\
d_1 & d_2 & d_n
\end{array} \]

\[ D \]

\[ \begin{array}{ccc}
0,1 & 0,1 & 0,1 \\
\vdots & \vdots & \vdots \\
0,1 & 0,1 & 0,1
\end{array} \]

\[ R \]
Implementing SIS Controller Component

SIS Controller

Swarm Membership (LA Action selection)

D1 D2 Dn

s1 sk

PSO Population

CV (Si, gbesti)

Swarm Refinement (LA Probability Update)

β1 β2

αn

βn

α2

α1

La1 La2 Lan
The Isolation of PSO populations

\[ \{p_0, p_1\} \]

\[ \text{Bi-population PSO} \]

\[ CV(S_i, g\text{best}_i) \]
Implementation

• Python 3.4 (we extensively use NumPy package)
• Github repository @ https://goo.gl/V0vTRM
• Code
  • PSO ~200 lines
  • AIO ~400 lines
• XML interface for reading configurations
• 5 benchmark functions
  • Sphere, Rosenbrock, Ackley, Griewanks, Rastirign
  • 30 dimensions
  • 50 particles
  • ...
Results – Sphere Benchmark
Results – Rosenbrock Benchmark
Results – Ackley Benchmark
Results – Griewanks Benchmark
Results – Rastrigin Benchmark

![Graph showing comparison between PSO and AIO iterations and fitness values. The graph demonstrates the decrease in fitness with increasing iteration for both algorithms. The y-axis represents fitness values ranging from $10^{-14}$ to $10^0$, and the x-axis represents iteration numbers from 1 to 10,000. The PSO and AIO curves are depicted with blue and black lines, respectively, illustrating the efficiency of AIO in achieving lower fitness values compared to PSO.]
Future Work

• Clustering
• Dimensionality reduction
• Feature selection
• Parameter estimation
Questions?