Nested Bilevel Evolutionary Algorithm (N-BLEA)

Ankur Sinha
Department of Information and Service Economy
Aalto University School of Business
Helsinki, 00076 Aalto, Finland
ankur.sinha@aalto.fi

Pekka Malo
Department of Information and Service Economy
Aalto University School of Business
Helsinki, 00076 Aalto, Finland
pekka.malo@aalto.fi

Kalyanmoy Deb
Department of Electrical and Computer Engineering
Michigan State University
East Lansing, MI 48823, USA
kdeb@egr.msu.edu

1 Introduction

Nested bilevel evolutionary algorithm is a nested strategy that uses evolutionary algorithm at both levels to handle bilevel optimization problems. The N-BLEA approach is intelligent such that it switches to quadratic programming at lower level whenever possible. It is able to handle lower dimensional SMD-Suite as well as the TP-Suite successfully. However, it being a nested approach, the function evaluations are high, particularly for the lower level. This approach has been used in [2, 3, 1] to handle bilevel test problems as well as real-world problems. The algorithm provided in this package is almost similar to what has been discussed in the above papers, but it differs slightly in the following ways:

1. QP is implemented at the lower level along with an evolutionary algorithm.
2. Utilizes different kind of termination criterion at both levels.
3. Lower level searches are performed with a reduced lower level population size in the intermediate upper level generations.

The above modifications help in faster execution of the algorithm. It increases its efficiency and in most of the cases produces better results than what is reported in the above mentioned papers. It should be noted that this is a naive strategy to handle bilevel problems. However, it may be useful for developers to use this approach as a benchmark to evaluate the performance of their algorithms.

2 Algorithm Description

In this section, we describe the nested bilevel evolutionary algorithm for single-objective bilevel optimization problems. The algorithm requires that a lower level optimization task be solved for every new set of upper level variables produced using the genetic operators. The method relies on a steady state single objective real coded genetic algorithm to solve the problems at both levels. The procedure has been discussed in [2, 3, 1]. However, the code provided in the N-BLEA package is slightly different from what has been discussed in the mentioned papers. Following is a step-by-step algorithm description for N-BLEA:

2.1 Upper Level Optimization Procedure

Step 1 *Initialization Scheme.* Initialize a random population (N_p) of upper level variables. For each upper level population member execute a lower level optimization procedure (Refer Sub-section 2.2) to determine the corresponding optimal lower level variables. Assign upper level fitness based on the upper level function value and constraints.

Step 2 *Selection of upper level parents.* Choose 2μ population members from the previous population and conduct a tournament selection to determine μ parents.

Step 3 *Evolution at the upper level.* Perform a PCX based crossover (Refer Sub-section 2.4) and a polynomial mutation to create λ offspring. This provides the upper level variables for each offspring.

Step 4 *Lower level optimization.* Solve the lower level optimization problem (Refer Sub-section 2.2) for each offspring. This provides the lower level variables for each offspring.

Step 5 *Evaluate offspring.* Combine the upper level variables with the corresponding optimal lower level variables for each offspring. Evaluate all the offspring based on upper level function value and constraints.

Step 6 *Population update.* Choose r random members from the parent population and pool them with the λ offspring. The best r members from the pool replace the chosen r members from the population.

Step 7 *Termination check.* Proceed to the next generation (*Step 2*) if the termination check (Refer Sub-section 2.6) is false.

2.2 Lower Level Optimization

At the lower level we first use a quadratic programming approach to find the optimum. If the procedure is unsuccessful we use a global optimization procedure using an evolutionary algorithm to find the optimum. The lower level optimization using evolutionary algorithm is able to handle complex optimization problems with multimodality. The fitness assignment at this level is performed based on lower level function value and constraints. Let the lower level population size be n_p. Note that n_p is reduced linearly with respect to upper level population variance, but it is not allowed to go below 6. Let the upper level member being optimized be $\mathbf{x}_u^{(0)}$. The steps for the lower level optimization procedure are as follows:

2.2.1 Lower level quadratic programming

Step 1 If execution is transferred from Step 1 of upper level then go to *Step a* otherwise go to *Step b*.

a: Create $\frac{(n_p+1)(n_p+2)}{2} + n_p + 1$ lower level points randomly. Evaluate all the points with respect to lower level objectives and constraints, and choose the best one as $\mathbf{x}_l^{(0)}$. Go to *Step 2*.

b: Determine the member closest to $\mathbf{x}_u^{(0)}$ in the upper level population. Denote the lower level optimal variables from the closest upper level member as $\mathbf{x}_l^{(0)}$. Create $\frac{(n_p+1)(n_p+2)}{2} + n_p$ lower level points about $\mathbf{x}_l^{(0)}$ using polynomial mutation. Go to *Step 2*.

Step 2 Construct a quadratic approximation for lower level objective function about $x_l^{(0)}$ using the created points. Construct linear approximations for the lower level constraints.

Step 3 Optimize the quadratic function with linear constraints using a sequentially quadratic programming approach.

Step 4 Compute the value of the optimum using the quadratic approximated objective function and the true lower level objective function. If the absolute difference is less than δ_{min} and the point is feasible and better than $\mathbf{x}_l^{(0)}$, accept the solution as lower level optimum, otherwise perform an evolutionary optimization search.

2.2.2 Lower level evolutionary optimization

Step 1 Quadratic programming is already executed but unsuccessful, therefore use an evolutionary approach to handle the lower level problem. Use the solution obtained using quadratic programming as one of the population members and randomly initialize other $n_p - 1$ lower level members. The upper level variables are kept fixed for all the population members. Assign lower level fitness based on the lower level function value and constraints.

Step 2 Choose 2μ members randomly from the lower level population. Perform a tournament selection with respect to lower level fitness to generate μ parents.

Step 3 Perform crossover and mutation to generate λ offspring.

Step 4 Evaluate each offspring with respect to lower level function and constraints.

Step 5 Choose r members randomly from the lower level population and pool them with the λ lower level offspring. The best r members with respect to lower level fitness replace the chosen r members from the lower level population.

Step 6 Proceed to the next generation (*Step 2*) if the termination check (Refer Sub-section 2.6) is false.

2.3 Parameters

The parameters in the algorithm were fixed as $\mu = 2$, $\lambda = 3$ and $r = 2$. Probability of crossover was fixed as 0.9 and the probability of mutation was fixed as 0.1. The crossover operator requires two parameters ω_ξ and ω_η, which are fixed as suggested in the next subsection.

2.4 Crossover Operator

The crossover operator used at both levels is a parent centric crossover operator. The operator creates an offspring from three parents, when one of the three parents is chosen as the index parent as follows,

$$\mathbf{c} = \mathbf{x_p} + \omega_\xi \mathbf{d} + \omega_\eta \frac{\mathbf{p_2} - \mathbf{p_1}}{2}. \quad (1)$$

The terms used in the above equation are defined as,

- $\mathbf{x_p}$ is the *index* parent
- $\mathbf{d} = \mathbf{x_p} - \mathbf{w}$, where $\mathbf{w}$ is the mean of μ parents
- $\mathbf{p_1}$ and $\mathbf{p_2}$ are the other two parents
- $\omega_\xi = 0.1$ and $\omega_\eta = \sum_{i=1}^{m_v} \frac{m_v}{|x_p^i - w^i|}$ are the two parameters, where $v \in \{u, l\}$ such that m_u is the number of variables at the upper level and m_l is the number of variables at the lower level.

The two parameters ω_ξ and ω_η, describe the extent of variations along the respective directions. While creating $\lambda = 2$ offspring from $\mu = 3$ parents, the best parent is chosen as an index parent everytime.

2.5 Constraint Handling

We define the constraint violation as the sum of violations of all the constraints at the respective levels. If a member at a particular level has a smaller constraint violation, then it is always preferred over a member with a higher constraint violation at the same level. A member with no constraint violation is deemed to be feasible, and is considered better than any of the other infeasible members. While comparing two feasible members, the member with a smaller function value at the level is preferred.

2.6 Termination Check

The algorithm uses a variance-based as well as improvement-based termination criteria at both levels. When the variance of the population members is below a particular threshold the optimization task is terminated. Further, in case the improvement in the function value of the elite is below the threshold after a fixed number of generation the optimization task terminates.

References

[1] A. Sinha, P. Malo, and K. Deb. Unconstrained scalable test problems for single-objective bilevel optimization. In *2012 IEEE Congress on Evolutionary Computation (CEC-2012)*. IEEE Press, 2012.

[2] A. Sinha, P. Malo, and K. Deb. Test problem construction for single-objective bilevel optimization. *Evolutionary Computation Journal*, 22(3):439–477, 2014.

[3] A. Sinha, P. Malo, A. Frantsev, and K. Deb. Finding optimal strategies in a multi-period multi-leader-follower stackelberg game using an evolutionary algorithm. *Computers & Operations Research*, 41:374–385, 2014.