A Bilevel Optimization Approach to Automated Parameter Tuning

Ankur Sinha, Pekka Malo, Peng Xu and Kalyanmoy Deb[*]

Department of Information and Service Economy
Aalto University School of Business
PO Box 21220, 00076 Aalto, Helsinki, Finland
Firstname.Lastname@aalto.fi

COIN Report Number: 2014013

May 12, 2014

Abstract

Many of the modern optimization algorithms contain a number of parameters that require tuning before the algorithm can be applied to a particular class of optimization problems. A proper choice of parameters may have a substantial effect on the accuracy and efficiency of the algorithm. Until recently, parameter tuning has mostly been performed using brute force strategies, such as grid search and random search. Guesses and insights about the algorithm are also used to find suitable parameters or suggest strategies to adjust them. More recent trends include the use of meta-optimization techniques. Most of these approaches are computationally expensive and do not scale when the number of parameters increases. In this paper, we propose that the parameter tuning problem is inherently a bilevel programming problem. Based on this insight, we introduce an evolutionary bilevel algorithm for parameter tuning. A few commonly used optimization algorithms (Differential Evolution and Nelder-Mead) have been chosen as test cases, whose parameters are tuned on a number of standard test problems. The bilevel approach is found to quickly converge towards the region of efficient parameters. The code for the proposed algorithm can be accessed from the website `http://bilevel.org`.

1 Introduction

When conventional solution procedures fail and exhaustive search is impractical, heuristic approaches like evolutionary algorithms are often used to optimize difficult problems. Over the past few decades, such approaches have been successful in solving a number of practical optimization problems. This has led to the design and development of problem-specific evolutionary algorithms to handle problems that are otherwise difficult to solve. Most of the heuristic procedures involve parameters that often require tuning during the design and development phase. The choice of parameter values has a significant influence on

[*]Kalyanmoy Deb is a Professor at the Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI, USA (Kdeb@egr.msu.edu).

the performance of the algorithms. The task is particularly challenging when the algorithm to be tuned is stochastic in nature.

Parameter tuning is a necessary step during the algorithm design phase. Though a number of strategies have been proposed, most of the techniques become computational very expensive when the number of parameters increases. Common strategies that researchers often use for parameter tuning are grid search and random search. These procedures are brute-force approaches that are applicable only when the number of parameters to be optimized is small. Researchers working on automated parameter tuning have proposed some structured search techniques based on grid search like Full factorial design [30]. Attempts have also been made towards more efficient techniques, such as model based methods [15, 34] and meta-evolutionary algorithms [17, 21]. Although considerable progress has been in this direction, there is still a significant need for more efficient strategies.

In this paper, we propose an evolutionary strategy for automated parameter tuning that can be used to tune both deterministic and stochastic optimization algorithms. The method differs from existing approaches as it draws insights from the domain of bilevel optimization. The contributions of the paper can be summarized as follows:

1. Formulating the automated parameter tuning task as a bilevel optimization problem
2. Proposing an algorithm based on bilevel optimization for automated parameter tuning
3. Suggesting a termination criterion for stochastic optimization problems.

The paper is organized as follows. In Section 2 we provide a literature survey of the parameter tuning approaches proposed in the past. We also discuss about the relevant studies on bilevel optimization. In Section 3 we provide a brief description of bilevel optimization problem. This is followed by the bilevel formulation of the parameter tuning problem in Section 4. In Section 5 we propose the algorithm based on the bilevel approach to handle parameter tuning problems. Thereafter, we create the test cases for evaluating our approach in Section 6, and provide the results of the approach in Section 7. Finally, we conclude the paper in Section 8.

2 Past Studies

In this paper, we show that the parameter tuning problem is inherently a bilevel optimization task. Although the problem of parameter tuning has been handled as a two layer optimization task, a direct link between parameter tuning and bilevel optimization has not been established. Therefore, we highlight the past studies on parameter tuning and bilevel optimization separately. This section has been divided into two parts. In the first part, we discuss the prior research on automated parameter tuning. In the second part, we highlight some of the past studies on evolutionary bilevel optimization.

2.1 Parameter Tuning

The parameter tuning problem has been widely studied and a number of different techniques have been proposed to find suitable parameters. In this section, we provide a brief survey of these techniques. For a detailed overview, we refer the readers to a study by Eiben and Smit [14], which also provides a conceptual framework for parameter tuning along with categorizations for the existing approaches.

One of the early approaches for parameter tuning is full factorial design, which is a straightforward technique for a comprehensive search. However, due to its computational expense it is often avoided and many of the new algorithms in the domain of evolutionary computing still rely on individual parameter-wise search to obtain suitable parameters. One way to reduce the computational expense in full factorial design is Graeco-latin square [30]. In this method, a few combinations are taken in order to allow a wide exploration in the factor range. Even though this dimensionality reduction diminishes the search effort, such a square still gets computationally expensive for larger number of parameters. Further, dimensionality reduction lowers the search potential by not accounting for possible interactions between parameters. Another method relying on a similar strategy is CALIBRA [1] that acts iteratively by refining and re-sampling in the area with better performance measures.

Model-based techniques are also used for computationally expensive problems. In these techniques, the target is to create a surrogate model that imitates the original problem. Since finding optimal parameters is a computationally expensive problem, a number of model-based techniques have been proposed. In these techniques a model is generated that approximates the performance measures for a given parameter vector without solving the corresponding optimization problem. Some of the model-based approaches are Least Square and Gradient Methods [22], Gaussian Models [15], and Logistic Regressions [34]. A natural extension of the technique would be to follow an iterative model construction to converge to the best parameter vector that has been proposed by Coy [9]. There also exist a number of screening methods [23, 7] that attempt to select the best parameter vector with as few simulations as possible.

Meta-evolutionary techniques have also been employed to optimize the parameters. These procedures follow a nested structure where an evolutionary algorithm at the meta-level acts on the parameters of another evolutionary algorithm that operates at a lower level. This is a classic example of a brute-force technique to solve bilevel optimization problems, and is a common strategy in the evolutionary bilevel optimization field. One of the first studies on meta-evolutionary algorithms was performed by Mercer and Sampson in 1978 [29] that optimized the genetic parameters. The study by Mercer and Sampson was preliminary and presented the results only from a single run. Greffenstette [17] did a more extensive study on meta-evolutionary algorithm by providing the experimental results from multiple runs of a genetic algorithm (GA). The performance of the GA was evaluated using two metrics, online and offline performances [12]. Some new extensions of the techniques in meta-optimization have been addressed in FocusedILS [21] and ParamILS [20].

2.2 Bilevel Optimization

We briefly highlight some of the studies in classical and evolutionary optimization literature on bilevel optimization. Bracken and Mcgill [6] introduced bilevel programming to the domain of mathematical programming in the early seventies. Since then a number of studies have been conducted on bilevel programming [8, 41, 13].

Researchers in the classical optimization community have attempted to solve bilevel programming problems using the Karush-Kuhn-Tucker approach [5, 19], branch-and-bound approach [4], and the use of penalty functions [2]. Commonly, the methodologies are suitable for problems adhering to simplifying assumptions of smoothness, linearity or convexity. One often needs to resort to other approaches like evolutionary techniques when bilevel problems get complex.

Many of the proposed evolutionary bilevel optimization algorithms are nested strategies [28, 45, 26, 39, 3], that is, they solve the lower level optimization problem completely for any given upper level decision. For instance, in [28, 45, 26] the upper level optimization task is handled using an evolutionary algorithm and the lower level is handled using a classical approach. In [39, 3] both levels are handled using an evolutionary technique. Researchers in the evolutionary community have also used the Karush-Kuhn-Tucker conditions at the lower level [42, 25, 27] to convert bilevel optimization into a single level constrained optimization task. Co-evolutionary approaches to handle bilevel optimization problems have been attempted in [32, 24]. Studies where mathematical insights have been successfully used with evolutionary algorithms to handle bilevel problems are [43, 44, 38]. Researchers have also worked on multi-objective extensions of bilevel optimization. Some of the studies in this direction are [18, 36, 11, 37, 35, 46].

3 A Brief Description of Bilevel Optimization

Bilevel optimization involves two levels of optimization tasks with one level nested within the other. The outer optimization task is commonly referred as the upper level optimization task, and the inner optimization task is commonly referred as lower level optimization task. The nested structure of the problem puts a constraint that only the optimal solutions to the lower level optimization task may be acceptable as possible feasible candidates to the upper level optimization task. There are two kinds of variables in a bilevel optimization problem: the upper level variables x_u, and the lower level variables x_l. Optimization at the lower level is performed with respect to the lower level variables x_l, while the upper level variables x_u act as parameters. An optimal lower level vector and the corresponding upper level vector x_u constitute a feasible upper level solution, provided that the pair also satisfies the upper level constraints. Below, we provide two equivalent formulations for a general bilevel optimization problem:

Definition 1 (Bilevel Optimization Problem) *Let $X = X_U \times X_L$ denote the product of the upper level decision space X_U and the lower level decision space X_L, i.e. $x = (x_u, x_l) \in X$, if $x_u \in X_U$ and $x_l \in X_L$. For the upper level objective function $F : X \to \mathbb{R}$ and the lower level objective function $f : X \to \mathbb{R}$, a general bilevel optimization problem is given by*

$$\begin{aligned} \underset{x \in X}{\mathit{Min}} \quad & F(x), \\ \textit{s.t.} \quad & x_l \in \underset{x_l \in X_L}{\operatorname{argmin}} \left\{ f(x) \mid g_i(x) \geq 0, i \in I \right\}, \\ & G_j(x) \geq 0, j \in J. \end{aligned} \tag{1}$$

where the functions $g_i : X \to \mathbb{R}$, $i \in I$, represent the lower level constraints and $G_j : X \to \mathbb{R}$, $j \in J$, is the collection of upper level constraints.

The above formulation requires that a member $x^{(0)} = (x_u^{(0)}, x_l^{(0)})$ can be considered feasible at the upper level if it satisfies the upper level constraints and $x_l^{(0)}$ be an optimal solution to the lower level problem corresponding to $x_u^{(0)}$. Therefore, in a bilevel program the lower level problem appears as a parameterized constraint to the upper level problem. Next, we provide an equivalent formulation of the bilevel optimization problem by replacing the lower level optimization problem with a set-valued function that maps the given upper level decision vector to the corresponding set of optimal lower level solutions.

Definition 2 (Alternative definition) *Let a set-valued function $\Psi : X_U \rightrightarrows X_L$, denote the optimal-solution set mapping of the lower level problem, i.e.*

$$\Psi(x_u) = \underset{x_l \in X_L}{\operatorname{argmin}} \left\{ f(x) \mid g_i(x) \geq 0, i \in I \right\}.$$

A general bilevel optimization problem (BLOP) is then given by

$$\begin{aligned} \underset{x \in X}{\mathit{Min}} \quad & F(x), \\ \textit{s.t.} \quad & x_l \in \Psi(x_u), \\ & G_j(x) \geq 0, j \in J. \end{aligned} \tag{2}$$

where the function Ψ may be a single-vector valued or a multi-vector valued function depending on whether the lower level function has multiple global optimal solutions or not.

4 Bilevel Formulation for Parameter Tuning

To the best knowledge of the authors, there does not exist a well-defined mathematical formulation of the parameter tuning problem. A close look at the bilevel optimization formulation reveals the "nested" connection between the parameter tuning problem and a bilevel optimization problem. Given this similarity, it is possible to write the parameter tuning problem as a bilevel program. Such formulation would allow bilevel solution approaches to be applicable for parameter tuning. Consider that $\mathcal{A}_{\mathcal{P}}$ represents an algorithm $\mathcal{A}$ operating on a problem $\mathcal{P}$ for which the algorithm parameters θ are supposed to be optimized. The performance of the algorithm is given by γ that follows an unknown distribution $\mathcal{D}_{\mathcal{A}_{\mathcal{P}}}$. The problem of finding the optimal set of parameters θ can be written as the following bilevel optimization problem:

$$\begin{aligned} \underset{\theta \in \Theta}{\text{minimize}} \quad & F(\gamma, \theta) = E[\gamma|\theta] \\ \text{subject to} \quad & \gamma = \mathcal{A}_{\mathcal{P}}(\theta), \end{aligned}$$

The problem $\mathcal{P}$ on which the algorithm $\mathcal{A}$ operates is given as follows:

$$\begin{aligned} \underset{z \in Z}{\text{minimize}} \quad & f^p(z) \\ \text{subject to} \quad & g^p_k(z) \leq 0, k \in K. \end{aligned}$$

where f_p represents an objective function to be optimized with respect to vector z subject to the constraints g^p_k. The above bilevel formulation for parameter tuning is a special case of Definition 1, where $x_u = \theta$ and $x_l = \gamma$.

5 Algorithm Description

In this section, we provide a description for the parameter tuning algorithm under the bilevel framework. The algorithm is motivated by a recently proposed bilevel evolutionary algorithm based on quadratic approximations (BLEAQ) [38] that bears similarities with the iterative model-based strategies attempted in the automated parameter tuning literature. In this paper we extend BLEAQ so that it can be applied to the problem of parameter tuning. The extended approach is referred as Bilevel Automated Parameter Tuning (BAPT) Algorithm.

The BAPT optimization strategy is based on quadratic approximation of the lower level optimal vector as a function of the upper level vector. In other words, the approach attempts to approximate the Ψ-mapping in the bilevel optimization case. However, the lower level optimization task in this case is noisy leading to different performance values in different runs of $\mathcal{A}_{\mathcal{P}}$ for a fixed θ. Therefore, multiple executions may be required for any given θ to be able to compute the expectation at the upper level. A high number of executions to compute the expectation might make the algorithm very expensive and a low number of executions would lead to a large noise causing problems in determining the fitness of solutions at the upper level. In the proposed BAPT approach we avoid large number of runs at the lower level and at the same time suppress noise arising from the lower level runs.

The operation of the BAPT algorithm can be summarized as follows. To begin with, we initialize an upper level population of size N with random θ_i, $i \in \{1, \dots, N\}$. For each θ_i, q lower level optimization runs of $\mathcal{A}_{\mathcal{P}}$ are executed leading to an array of performance values given as $\gamma_{i,j}, j \in \{1, \dots, q\}$. Let these performance values[1] be collectively represented as γ_i. An expectation over $\gamma_{i,j}$ ($F_i = \frac{1}{q}\Sigma_{j=1}^{q}\gamma_{i,j}$) leads to the upper level objective value for (θ_i, γ_i). Based on these results we construct a quadratic approximation of F (or $E[\gamma|\theta]$) using θ. The approximation step serves a dual purpose in the algorithm. First, the quadratic approximation is used to determine F (or $E[\gamma|\theta]$) for θ when $\mathcal{A}_{\mathcal{P}}$ is not executed leading to savings in the executions of $\mathcal{A}_{\mathcal{P}}$. Second, the quadratic approximation is used to assign fitness values (Fit_i) to upper level population members that leads to a reduction in noise. Below we provide a step-by-step procedure for the BAPT algorithm.

S. 1 Initialize a random population of θ_i vectors of size N. For each $\theta_i, i \in \{1, \dots, N\}$ execute $\mathcal{A}_{\mathcal{P}}$ q times and store $\gamma_{i,j}, j \in \{1, \dots, q\}$.

S. 2 Compute upper level function $F(\gamma_i; \theta_i) = \frac{1}{q}\Sigma_{j=1}^{q}\gamma_{i,j}$ value for all $i \in \{1, \dots, N\}$. Tag all the population members as 1.

S. 3 Copy all the population members (θ_i, γ_i), $i \in \{1, \dots, N\}$ into an archive A.

S. 4 Perform a localized quadratic approximation of F (or $E[\gamma|\theta]$) over θ about each population member by choosing $\frac{(dim(\theta)+1)(dim(\theta)+2)}{2} + dim(\theta)$ closest members [2] from the archive A. Use the value of the quadratic function to assign fitness to the respective population and archive members.

S. 5 Given the current population, choose the best member by fitness as one of the parents. Choose $\mu - 1$ more parents by performing a tournament selection among $2(\mu-1)$ members randomly chosen from the population.

S. 6 Perform crossover among the chosen parents by using the best parent as an index parent. Create λ offspring from the cross-over operation.

S. 7 Assign fitness to the λ offspring using a quadratic approximation in a similar manner as in Step 4..

S. 8 If the number of tag 1 members in the population is less than $\frac{N}{2}$, then execute $\mathcal{A}_{\mathcal{P}}$ q times on the offspring and compute the objective function value for the offspring; tag the offspring as 1 and copy them to the archive. If the number of tag 1 members in the population is greater than or equal to $\frac{N}{2}$ then assign fitness from quadratic approximation as the objective function value for the offspring; tag the offspring as 0 and do not copy them to the archive.

S. 9 Choose r members from the parent population. Form a pool of chosen r members and λ offspring. The best r members (in terms of fitness) from the pool replace the chosen r members from the population. A termination check (refer Subsection 5.4) is performed and the algorithm moves to the next generation (Step 4) if the termination check is false.

[1] In the later part of this paper, we will propose $q = 1$, thus producing a single performance value for a given θ_i. This avoids multiple computations at the lower level for a given θ_i leading to savings in computation.

[2] At least $\frac{(dim(\theta)+1)(dim(\theta)+2)}{2}$ points are required for a quadratic approximation. Extra points are chosen to avoid over-fitting.

Table 1: Description of the chosen algorithms and their parameters to be optimized. The bounds chosen for the parameters to be optimized in this study are also provided.

Algo.	Description	Parameters to be optimized
DE	Differential evolution is a population based search heuristic that uses vector differences to proceed towards the optimum. In differential evolution, at each generation of the algorithm a mutant vector is constructed for different members in the population. In a simple DE algorithm, the mutation operation is based on adding differences between randomly selected members to another member, though other variants are also possible. This is followed by a recombination operation where a trial vector is constructed using the initial member and the mutant vector. The trial member competes with the current population members in order to enter the population. A number of variants are possible for DE by varying the mutation and the recombination operation. In this paper, we use DE/rand/1/bin variant, which means that the mutation vector is constructed by choosing the members randomly from the population, a single difference is used to arrive at the mutant vector, and the crossover type is binomial.	Differential Weight: $F \in [0, 2]$ Crossover Probability: $CR \in [0, 1]$
NM	Nelder-Mead method is a non-linear derivative-free optimization technique to find the local minimum of a function. For a function with d dimensions or variables, the method starts with a simplex that is a polytope with $d + 1$ vertices in the hyperspace. It is a heuristic based on pattern search that compares the function values at the vertices and updates the simplex. The method involves the steps of reflection, expansion and contraction to converge towards a minimum. The simplex gets updated at each of these steps based on the respective coefficients that appear as parameters in the Nelder-Mead algorithm. We use the basic version of the Nelder-Mead algorithm proposed in [31] in our computations.	Reflection Coefficient: $\alpha \in [0, 2]$ Expansion Coefficient: $\gamma \in [1, 4]$ Contraction Coefficient: $\rho \in [0, 1]$ Shrink Coefficient, σ, has not been considered in this study.

5.1 Constraint handling

In the current setting, there are no equality or inequality constraints. The formulation contains only box constraints that are taken care by assigning the boundary value if one or more of the upper level variables (θ) violate it. However, there could be situations when the user might like to have a functional dependence between two or more variables at the upper level. This could give rise to equality or inequality constraints. In such situations we use the constraint handling procedure proposed in [10]. The overall constraint violation for any solution is the summation of the violations of all the equality and inequality constraints. A solution $x^{(i)}$ is said to 'constraint dominate' a solution $x^{(j)}$ if any of the following conditions are true:

1. Solution $x^{(i)}$ is feasible and solution $x^{(j)}$ is not.
2. Solution $x^{(i)}$ and $x^{(j)}$ are both infeasible but solution $x^{(i)}$ has a smaller overall constraint violation.
3. Solution $x^{(i)}$ and $x^{(j)}$ are both feasible but the objective value of $x^{(i)}$ is less than that of x^{j}.

5.2 Parameters

The parameters in the algorithm are fixed as $\mu = 3$, $\lambda = 2$ and $r = 2$. The crossover and mutation probabilities are fixed as 0.9 and 0.1 respectively. The upper level population size N is fixed as 50 and the number of lower level executions q for a given parameter vector is fixed as 1.

5.3 Crossover Operator

The crossover operator used in the algorithm is similar to the PCX operator proposed in [40] with minor modifications. The operator creates a new solution from $\mu = 3$ parents as follows:

$$c = x^{(p)} + \omega_\xi d + \omega_\eta \frac{p^{(2)} - p^{(1)}}{2} \tag{3}$$

The terms used in the above equation are defined as follows:

- $x^{(p)}$ is the *index* parent which is always the best parent in this algorithm
- $d = x^{(p)} - g$, where g is the mean of μ parents
- $p^{(1)}$ and $p^{(2)}$ are the other two parents
- $\omega_\xi = 0.1$ and $\omega_\eta = \frac{dim(x^{(p)})}{||x^{(p)} - g||_1}$ are the two parameters.

Parameters ω_ξ and ω_η, describe the extent of variations along the respective directions.

5.4 Termination

In this paper, we suggest a termination criterion inspired by [16], which is commonly used with Markov chain Monte Carlo (MCMC) algorithms. The criterion is used to determine the stationarity of a time variant chain. It is noteworthy that the stochasticity at the lower level causes the mean of the population at the upper level to vary from one generation to the other. At every generation of the algorithm we extract the population means (say θ^m) from the past g_l generations. The criterion is based on the analysis of two parts (usually start and end) of the extracted chain. A z-score from two parts of the chain, say p_1 and p_2, is computed as follows:

$$z_{12} = \frac{\bar{\theta}^m_{p_1} - \bar{\theta}^m_{p_2}}{\sqrt{Var(\bar{\theta}^m_{p_1}) + Var(\bar{\theta}^m_{p_2})}}$$

Note that the terms in the numerator represent the means of the two parts of the chain while the terms in the denominator represent the asymptotic variances of $\bar{\theta}^m_{p_1}$ and $\bar{\theta}^m_{p_2}$. The asymptotic variances are estimated from the spectral density at zero.

In this paper we choose $g_l = 100$ and divide the chain as $p_1 = (1, 10), p_2 = (11, 20), p_3 = (21, 30), p_4 = (31, 40), p_5 = (41, 50)$ and $p_6 = (51, 100)$. Five different z-scores, $z_{16}, z_{26}, z_{36}, z_{46}$ and z_{56} are computed, and if all the absolute z-scores are smaller than 0.0125 then we terminate the algorithm.

6 Test Cases

To evaluate the BAPT procedure, we construct 2 different test cases. Each test case represents an algorithm whose parameters we want to optimize. Within each test case there are 8 test problems for which we optimize the parameters of the algorithms. The algorithms that we have used in this paper are: Differential Evolution (DE) [33] and Nelder-Mead (NM) [31]. Table 1 provides a brief description for the basic versions of the algorithms and their parameters that are optimized using the BAPT approach on the chosen set of 8 test problems. The test problems on which the parameters of these algorithms are optimized are given in Table 2. The chosen test problems are small dimensional unconstrained optimization tasks.

7 Results

In this section, we present the results obtained from BAPT for the two test cases i.e. DE and NM. For each test case, when executed on a specific test problem, we provide the mean and the standard deviation for the optimal parameter values obtained from 21 runs of BAPT. To evaluate the algorithm performance we consider grid search as the baseline. At each grid point we perform 101 optimization runs with DE and NM for every test problem. We provide the results for the test cases separately in the following sub-sections.

7.1 DE Results

The results of the BAPT approach, when used to optimize the parameters of differential evolution, are presented in Table 3 for individual test problems. The table shows the mean value and the standard deviation of the best parameters obtained from 21 runs of BAPT for each test problem. The small standard deviations suggest that the BAPT approach converges in the same region in most of its runs. The results have been contrasted with the grid search approach that has an accuracy of $(\pm 0.1, \pm 0.1)$. It can be seen that the mean result obtained from the algorithm lies close to the result obtained from grid search. To give a better idea of the performance of BAPT, we have plotted the best parameter values obtained from 21 runs for Sphere and Schwefel functions in Figures 1 and 2 respectively. The contours in the figures represent the average performance of the parameters obtained using a fine-grained grid search at equally spaced points in the 2-D plane. The dark region represents better performaing parameters and the lighter region represents worse performing parameters. It can be observed that the good parameter region is a flat valley suggesting a range of possible parameters that would perform well on the problems. Our approach converges close to the optimal region in all of the 21 runs.

Table 2: Description of the selected test problems (TP1-TP8).

Problem	Formulation	Optimal Solution
TP1 (Sphere) $dim(z) = 2$	$\underset{z}{\text{Min}}\ f^p(z) = z_1^2 + z_2^2.$	$f^p = 0.0, z = (0.0, 0.0).$
TP2 (Himmelblau) $dim(z) = 2$	$\underset{z}{\text{Min}}\ f^p(z) = (z_1^2 + z_2 - 11)^2 + (z_1 + z_2^2 - 7)^2.$	$f^p = 0.0, z = (3.0, 3.0),$ $f^p = 0.0, z = (3.58439, -1.84813),$ $f^p = 0.0, z = (-3.77934, -3.28317),$ $f^p = 0.0, z = (-2.80512, 3.13134).$
TP3 (Schwefel) $dim(z) = 2$	$\underset{z}{\text{Min}}\ f^p(z) = \Sigma_{i=1}^2 \big(\Sigma_{j=1}^i z_j\big)^2.$	$f^p = 0.0, z = (1.0, 1.0).$
TP4 (Bohachevsky) $dim(z) = 2$	$\underset{z}{\text{Min}}\ f^p(z) = z_1^2 - 0.3\cos(3\pi z_1) + 2z_2^2 - 0.4\cos(4\pi z_2) + 0.7.$	$f^p = 0.0, z = (0.0, 0.0).$
TP5 (Goldstein-Price) $dim(z) = 2$	$\underset{z}{\text{Min}}\ f^p(z) = \big(1 + a(z)^2 b(z)\big)\big(30 + c(z)^2 d(z)\big),$ where $a(z) = z_1 + z_2 + 1.0,$ $b(z) = 19 - 14z_1 + 3z_1^2 - 14z_2 + 6.0z_1z_2 + 3z_2^2,$ $c(z) = 2z_1 - 3z_2,$ $d(z) = 18 - 32z_1 + 12z_1^2 + 48z_2 - 36z_1z_2 + 27z_2^2.$	$f^p = 3.0, z = (0.0, -1.0).$
TP6 (Mckinnon) $dim(z) = 2$	$\underset{z}{\text{Min}}\ f^p(z) = \left\{\begin{matrix} \alpha\beta\lvert z_1\rvert^\tau + z_2(1.0 + z_2) & z_1 \leq 0 \\ \alpha z_1^\tau + z_2(1.0 + z_2) & z_1 > 0 \end{matrix}\right\}.$ where $\alpha = 6, \tau = 2, \beta = 60.$	$f^p = -0.25, z = (0.0, -0.5).$
TP7 (Beale) $dim(z) = 2$	$\underset{z}{\text{Min}}\ f^p(z) = \big(1.5 - z_1(1 - z_2)\big)^2 + \big(2.25 - z_1(1 - z_2^2)\big)^2 + \big(2.625 - z_1(1 - z_2^3)\big)^2$	$f^p = 0.0, z = (3.0, 0.5).$
TP8 (Powell) $dim(z) = 4$	$\underset{z}{\text{Min}}\ f^p(z) = \big(z_1 + 10z_2\big)^2 + \big(z_3 - z_4\big)^2 + \big(z_2 - 2z_3\big)^2 + \big(z_1 - z_4\big)^2.$	$f^p = 0.0, z = (0.0, 0.0, 0.0, 0.0).$

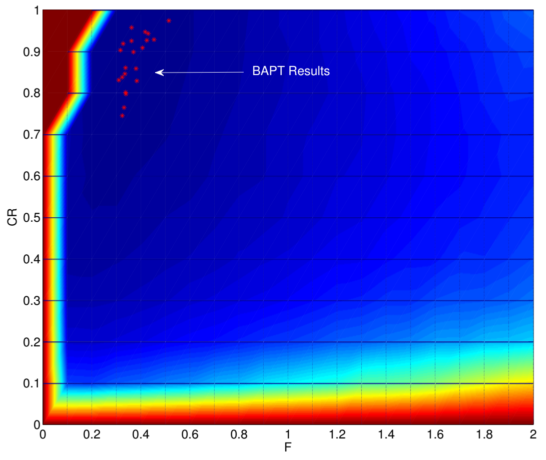

Figure 1: Performance of different DE parameters obtained using grid search is shown with contours when the DE is executed on sphere. The good parameters obtained using BAPT from 21 runs for DE-Sphere are shown as points in the flat valley.

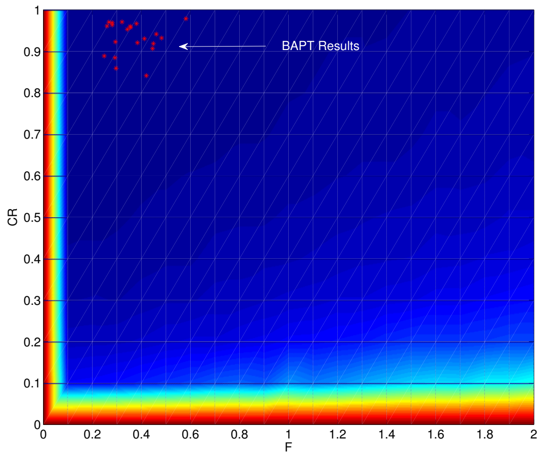

Figure 2: Performance of different DE parameters obtained using grid search is shown with contours when the DE is executed on schwefel. The good parameters obtained using BAPT from 21 runs for DE-Schwefel are shown as points in the flat valley.

7.2 NM Results

Nelder-Mead algorithm requires a few starting points equal to one more than the number of dimensions of the test problem being solved. In the experimental runs we initialized the starting points randomly in a cubic polyhedron (size 1) about the optimum. The results of the BAPT approach for Nelder-Mead algorithm are presented in Table 4. For this test case as well the standard deviations suggest that the BAPT algorithm converges with respect to α, γ and ρ. Once again the optimal landscape is expected to be a stochastic flat valley. The results have once again been compared with grid search. The accuracy of grid search in this case is $(\pm 0.2, \pm 0.2, \pm 0.2)$. The number of lower level calls required by BAPT to optimize DE and NM on respective test problems is given in Table 5.

7.3 Algorithm Analysis

One of the important aspects to be analyzed is the number of lower level runs, q, that should be performed for any given parameter vector. Keeping all the other aspects in the algorithm fixed, we varied q as $\{1, 3, 5, 10, 20\}$ for DE-Sphere and DE-Schwefel in order to determine the appropriate number of lower level runs that should be performed for a given parameter vector. For each q we determined the accuracy (with respect to fined grained grid search) and total number of lower level optimization calls required to optimize the problem. The normalized average results are given in Figure 3, where surprisingly $q = 1$ performed significantly better than any other q. One of the reasons for this is that performing a large number of runs at a single point in a region provides the algorithm an accurate performance estimate at that point. However, performing the same number of runs at different points in the same region may lead to more information, i.e. it provides an approximate performance estimate for the region and it also provides an indication to the algorithm of the direction in which further search should be performed. For example,

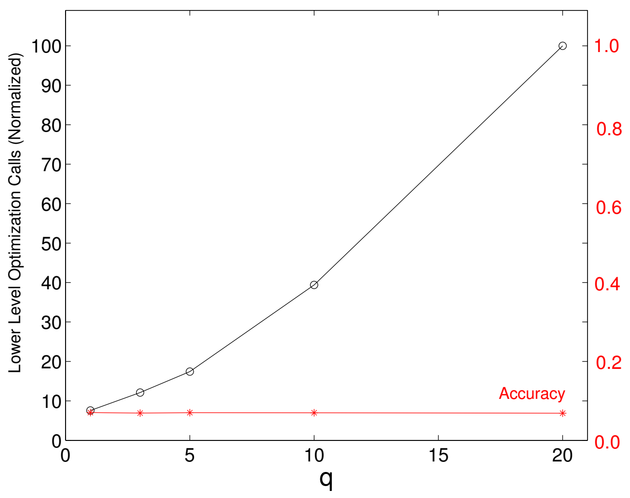

Figure 3: Computational expense and accuracy (distance from the best grid search point) against number of lower level runs q for a given parameter vector.

at the start of the algorithm, if a number of parameter vectors in a particular region perform poorly in single runs, then the region is quickly eliminated without wasting computational effort for computing accurate performance values.

8 Conclusions

In this paper, we have formulated the parameter tuning problem as a stochastic bilevel optimization problem. An algorithm (BAPT) has been proposed that draws ideas from the domain of bilevel optimization and automated parameter tuning to solve the parameter tuning problem. A new termination criterion is also suggested that can be used for evolutionary algorithms operating on stochastic problems. The results using the BAPT approach has been provided on two test cases and have been contrasted with grid search. The algorithm is found to converge close to the best parameter for most of the cases.

References

[1] B. Adenso-Diaz and M. Laguna. Fine-tuning of algorithms using factional experimental designs and local search. *Operations Research*, 54:99–114, 2006.

[2] E. Aiyoshi and K. Shimizu. Hierarchical decentralized systems and its new solution by a barrier method. *IEEE Transactions on Systems, Man, and Cybernetics*, 11:444–449, 1981.

[3] J. Angelo, E. Krempser, and H. Barbosa. Differential evolution for bilevel programming. In *Proceedings of the 2013 Congress on Evolutionary Computation (CEC-2013)*. IEEE Press, 2013.

[4] J. Bard and J. Falk. An explicit solution to the multi-level programming problem. *Computers and Operations Research*, 9:77–100, 1982.

Table 3: Parameter values obtained using BAPT and grid search for Differential Evolution.

	Differential Evolution				Grid Search	
	mean		std.			
TP	F	CR	F	CR	F	CR
1	0.3703	0.8760	0.0531	0.0657	0.4	1
2	0.3903	0.9199	0.0472	0.0611	0.4	1
3	0.3622	0.9331	0.0880	0.0389	0.3	1
4	0.3502	0.8584	0.0236	0.0693	0.4	0.8
5	0.4168	0.9586	0.0559	0.0290	0.5	0.9
6	0.4537	0.8627	0.0525	0.0671	0.4	0.9
7	0.4544	0.9488	0.0620	0.0347	0.5	1
8	0.5109	0.9142	0.0405	0.0487	0.5	0.9

Table 4: Parameter values obtained using BAPT and grid search for Nelder-Mead.

	Nelder-Mead						Grid Search Results		
	mean			std.					
TP	α	γ	ρ	α	γ	ρ	α	γ	ρ
1	1.1627	2.6741	0.3984	0.1958	0.2582	0.0474	1.0	2.8	0.4
2	1.1053	2.7099	0.4086	0.1493	0.3272	0.0352	1.0	3.0	0.4
3	1.0238	2.5320	0.3233	0.1082	0.3484	0.0830	1.2	2.0	0.2
4	1.4371	2.7394	0.5249	0.1550	0.4335	0.0677	1.0	2.8	0.4
5	1.3602	2.4991	0.4896	0.1443	0.3673	0.0743	1.2	3.8	0.4
6	0.9580	2.6641	0.4483	0.1161	0.3699	0.0543	0.8	2.8	0.4
7	1.1833	2.5826	0.4271	0.1612	0.3797	0.0663	1.0	2.2	0.4
8	1.0380	2.4765	0.5223	0.0546	0.3251	0.0461	1.0	2.4	0.4

Table 5: Number of lower level optimization calls required by the BAPT approach.

TP	Differential Evolution			Nelder-Mead		
	min	median	max	min	median	max
1	360	588	1034	500	790	1088
2	374	634	1000	486	752	960
3	380	494	996	458	696	1026
4	310	572	944	914	1038	1192
5	372	740	1096	488	936	1172
6	334	632	1062	466	778	1068
7	480	906	1104	538	638	1054
8	340	620	1008	294	790	1128

[5] L. Bianco, M. Caramia, and S. Giordani. A bilevel flow model for hazmat transportation network design. *Transportation Research Part C: Emerging technologies*, 17(2):175–196, 2009.

[6] J. Bracken and J. McGill. Mathematical programs with optimization problems in the constraints. *Operations Research*, 21:37–44, 1973.

[7] J. Branke, E. Chick, and C. Schmidt. New developments in ranking and selection: an empirical comparison of the three main approaches. In *WSC 2005: Proceedings of the 37th conference on Winter simulation*, pages 708–717. Winter Simulation Conference, 2005.

[8] B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of Operational Research*, 153:235–256, 2007.

[9] S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7:77–97, 2001.

[10] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4):311–338, 2000.

[11] K. Deb and A. Sinha. An efficient and accurate solution methodology for bilevel multi-objective programming problems using a hybrid evolutionary-local-search algorithm. *Evolutionary Computation Journal*, 18(3):403–449, 2010.

[12] K. A. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, Ann Arbor, MI: University of Michigan, 1975. Dissertation Abstracts International 36(10), 5140B (University Microfilms No. 76-9381).

[13] S. Dempe, J. Dutta, and S. Lohse. Optimality conditions for bilevel programming problems. *Optimization*, 55(5–6):505–524, 2006.

[14] A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1:19–31, 2011.

[15] M. El-Beltagy, P. Nair, and A. Keane. Metamodeling techniques for evolutionary optimization of computationally expensive problems: promises and limitations. In W. Banshaf, J. Daida, E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 196–203. Morgan Kaufmann, 1999.

[16] J. Geweke. Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments. *In Bayesian Statistics 4 (eds. J.M. Bernardo, J. Berger, A.P. Dawid and A.F.M. Smith)*, pages 169–193, 1992.

[17] J. Greffenstette. Optimisation of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 16:122–128, 1986.

[18] W. Halter and S. Mostaghim. Bilevel optimization of multi-component chemical systems using particle swarm optimization. In *Proceedings of World Congress on Computational Intelligence (WCCI-2006)*, pages 1240–1247, 2006.

[19] J. Herskovits, A. Leontiev, G. Dias, and G. Santos. Contact shape optimization: A bilevel programming approach. *Struct Multidisc Optimization*, 20:214–221, 2000.

[20] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützel. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[21] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *IN AAAI 2007: Proc. of the Twenty-Second Conference on Artificial Intelligence*, pages 1152–1157, 2007.

[22] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9:3–12, 2005.

[23] S.-H. Kim and B. L. Nelson. A fully sequential procedure for indifference-zone selection in simulation. *ACM Transactions on Modeling and Computer Simulation*, 11:251–273, 2001.

[24] F. Legillon, A. Liefooghe, and E.-G. Talbi. Cobra: A cooperative coevolutionary algorithm for bi-level optimization. In *2012 IEEE Congress on Evolutionary Computation (CEC-2012)*. IEEE Press, 2012.

[25] H. Li and Y. Wang. A genetic algorithm for solving a special class of nonlinear bilevel programming problems. In Y. Shi, G. Albada, J. Dongarra, and P. Sloot, editors, *Computational Science AI ICCS 2007*, volume 4490 of *Lecture Notes in Computer Science*, pages 1159–1162. Springer Berlin Heidelberg, 2007.

[26] H. Li and Y. Wang. A hybrid genetic algorithm for solving nonlinear bilevel programming problems based on the simplex method. *International Conference on Natural Computation*, 4:91–95, 2007.

[27] H. Li and Y. Wang. An evolutionary algorithm with local search for convex quadratic bilevel programming problems. *Applied Mathematics and Information Sciences*, 5(2):139–146, 2011.

[28] R. Mathieu, L. Pittard, and G. Anandalingam. Genetic algorithm based approach to bi-level linear programming. *Operations Research*, 28(1):1–21, 1994.

[29] R. Mercer and J. Sampson. Adaptive search using a reproductive metaplan. *Kybernetes*, 7:215–228, 1978.

[30] R. Myers and E. R. Hancock. Empirical modelling of genetic algorithms. *Evolutionary Computation*, 9:461–493, 2001.

[31] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[32] V. Oduguwa and R. Roy. Bi-level optimization using genetic algorithm. In *Proceedings of the 2002 IEEE International Conference on Artificial Intelligence Systems (ICAIS 2002)*, pages 322–327, 2002.

[33] K. V. Price, R. Storn, and J. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer-Verlag, Berlin, 2005.

[34] I. Ramos, M. Goldbarg, E. Goldbarg, and A. Neto. Logistic regression for parameter tuning on evolutionary computation. In *IEEE Congress on Evolutionary Computation*, Edinburgh, UK, 2005. IEEE Press.

[35] S. Ruuska and K. Miettinen. Constructing evolutionary algorithms for bilevel multiobjective optimization. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–7, june 2012.

[36] X. Shi. and H. S. Xia. Model and interactive algorithm of bi-level multi-objective decision-making with multiple interconnected decision makers. *Journal of Multi-Criteria Decision Analysis*, 10(1):27–34, 2001.

[37] A. Sinha and K. Deb. Towards understanding evolutionary bilevel multi-objective optimization algorithm. In *IFAC Workshop on Control Applications of Optimization (IFAC-2009)*, volume 7. Elsevier, 2009.

[38] A. Sinha, P. Malo, and K. Deb. Efficient evolutionary algorithm for single-objective bilevel optimization. *CoRR*, abs/1303.3901, 2013.

[39] 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.

[40] A. Sinha, A. Srinivasan, and K. Deb. A population-based, parent centric procedure for constrained real-parameter optimization. In *2006 IEEE Congress on Evolutionary Computation (CEC-2006)*, pages 239–245. IEEE Press, 2006.

[41] L. N. Vicente and P. H. Calamai. Bilevel and multilevel programming: A bibliography review. *Journal of Global Optimization*, 5(3):291–306, 2004.

[42] G. Wang, Z. Wan, X. Wang, and Y. Lv. Genetic algorithm based on simplex method for solving linear-quadratic bilevel programming problem. *Comput Math Appl*, 56(10):2550–2555, 2008.

[43] Y. Wang, Y. C. Jiao, and H. Li. An evolutionary algorithm for solving nonlinear bilevel programming based on a new constraint-handling scheme. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 32(2):221–232, 2005.

[44] Y. Wang, H. Li, and C. Dang. A new evolutionary algorithm for a class of nonlinear bilevel programming problems and its global convergence. *INFORMS Journal on Computing*, 23(4):618–629, 2011.

[45] Y. Yin. Genetic algorithm based approach for bilevel programming models. *Journal of Transportation Engineering*, 126(2):115–120, 2000.

[46] T. Zhang, T. Hu, Y. Zheng, and X. Guo. An improved particle swarm optimization for solving bilevel multiobjective programming problem. *Journal of Applied Mathematics*, 2012.