12 A Framework for Self-Tuning Algorithms

The performance of any algorithm will largely depend on the setting of its algorithm-dependent parameters. The optimal setting should allow the algorithm to achieve the best performance for solving a range of optimization problems. However, such parameter tuning is itself a tough optimization problem. In this chapter, we present a framework for self-tuning algorithms so that an algorithm to be tuned can be used to tune the algorithm itself.

12.1 Introduction

Since all algorithms have algorithm-dependent parameters, the performance of an algorithm largely depends on the values or setting of these parameters. Ideally, there should be a good way to tune these parameters so that the performance of the algorithm can be optimal in the sense that the algorithm can find the optimal solution of a problem using the minimal number of iterations and with the highest accuracy. However, such tuning of algorithm-dependent parameters is itself a very tough optimization problem. In essence, it is a hyperoptimization problem, that is, the optimization of optimization. In fact, finding the best parameter setting of an algorithm is still an open problem.

There are studies on parameter tuning. For example, Eiben provided a comprehensive summary of existing studies [2]. However, these studies are still very preliminary. There is no method of self-tuning in algorithms. Therefore, the main objective of this chapter is to provide a framework for self-tuning algorithms so that an algorithm can be used to tune its own parameters automatically, based on the self-tuning framework developed by Yang et al. [8]. As far as we are concerned, this is the first of its kind in parameter tuning.

12.2 Algorithm Analysis and Parameter Tuning

An optimization algorithm is essentially an iterative procedure, starting with some initial guess point/solution with an aim to reach a better solution or ideally the optimal solution to a problem of interest. This process of searching for optimality is generic, though the details of the process can vary from algorithm to algorithm. Traditional algorithms such as Newton-Raphson methods use a deterministic trajectory-based method,
whereas modern nature-inspired algorithms often are population-based algorithms that use multiple agents. In essence, these multiple agents form an iterative, dynamic system that should have some attractors or stable states. On the other hand, the same system can be considered a set of Markov chains so that they will converge toward some stable probability distribution.

12.2.1 A General Formula for Algorithms

Whatever the perspective, the aim of such an iterative process is to let the system evolve and converge into some stable optimality. In this case, it has strong similarity to a self-organizing system. Such an iterative, self-organizing system can evolve according to a set of rules or mathematical equations. As a result, such a complex system can interact and self-organize into certain converged states, showing some emergent characteristics of self-organization. In this sense, the proper design of an efficient optimization algorithm is equivalent to finding efficient ways to mimic the evolution of a self-organizing system [1,3].

From a mathematical point of view, an algorithm $A$ tends to generate a new and better solution $x^{t+1}$ to a given problem from the current solution $x^t$ at iteration or time $t$. In modern metaheuristic algorithms, randomization is often used in an algorithm, and in many cases randomization appears in the form of a set of $m$ random variables $\mathbf{\epsilon} = (\epsilon_1, \ldots , \epsilon_m)$ in an algorithm. For example, in simulated annealing, there is one random variable, whereas in particle swarm optimization, there are two random variables. In addition, there is often a set of $k$ parameters in an algorithm. For example, in particle swarm optimization, there are four parameters (two learning parameters, one inertia weight, and the population size). In general, we can have a vector of parameters $\mathbf{p} = (p_1, \ldots , p_k)$. Mathematically speaking, we can write an algorithm with $k$ parameters and $m$ random variables as

$$x^{t+1} = A \left( x^t, p(t), \mathbf{\epsilon}(t) \right),$$

where $A$ is a nonlinear mapping from a given solution (a $d$-dimensional vector $x^t$) to a new solution vector $x^{t+1}$.

12.2.2 Type of Optimality

Representation (12.1) gives rise to two types of optimality: optimality of a problem and optimality of an algorithm. For an optimization problem such as $\min f(x)$, there is a global optimal solution, whatever the algorithmic tool we may use to find this optimality. This is the optimality for the optimization problem. On the other hand, for a given problem $\Phi$ with an objective function $f(x)$, there are many algorithms that can solve it. Some algorithms may require less computational effort than others. There may be the best algorithm with the least computing cost, though this may not be unique. However, this is not our concern here. Once we have chosen an algorithm $A$ to solve a problem $\Phi$, there is an optimal parameter setting for this algorithm so that it can achieve the best performance. This optimality depends on both the algorithm itself and the problem it solves. In the rest of this chapter, we focus on this type of optimality.
That is, the optimality to be achieved is

\[
\text{Maximize the performance of } \xi = A(\Phi, p, \varepsilon)
\]  

(12.2)

for a given problem \( \Phi \) and a chosen algorithm \( A(., p, \varepsilon) \). We denote this optimality as \( \xi_\ast = A_\ast(\Phi, p_\ast) = \xi(\Phi, p_\ast) \), where \( p_\ast \) is the optimal parameter setting for this algorithm so that its performance is the best. Here we have used a fact that \( \varepsilon \) is a random vector that can be drawn from some known probability distributions. Thus the randomness vector should not be related to the algorithm optimality.

It is worth pointing out that there is another potential optimality. That is, for a given problem, a chosen algorithm with the best parameter setting \( p_\ast \), we can still use different random numbers drawn from various probability distributions and even chaotic maps so that even better performance may be achieved. Strictly speaking, if an algorithm \( A(., ., \varepsilon) \) has a random vector \( \varepsilon \) that is drawn from a uniform distribution \( \varepsilon_1 \sim U(0, 1) \) or from a Gaussian \( \varepsilon_2 \sim N(0, 1) \), it becomes two algorithms \( A_1 = A(., ., \varepsilon_1) \) and \( A_2 = A(., ., \varepsilon_2) \). Technically speaking, we should treat them as different algorithms. Since our emphasis here is about parameter tuning to find the optimal setting of parameters, we omit the effect of randomness vectors and thus focus on

\[
\text{Maximize } \xi = A(\Phi, p).
\]  

(12.3)

In essence, tuning an algorithm involves tuning its algorithm-dependent parameters. Therefore, parameter tuning is equivalent to algorithm tuning in the present context.

### 12.2.3 Parameter Tuning

To tune \( A(\Phi, p) \) so as to achieve its best performance, a parameter-tuning tool, i.e., a tuner, is needed. As with tuning high-precision machinery, sophisticated tools are required. For tuning parameters in an algorithm, what tool can we use? One way is to use a better, existing tool (say, algorithm \( B \)) to tune an algorithm \( A \). Now the question may become: How do you know \( B \) is better? Is \( B \) well tuned? If yes, how do you tune \( B \) in the first place? Naively, if we use another tool (say, algorithm \( C \)) to tune \( B \). Now again the question becomes, how has algorithm \( C \) been tuned? This can go on and on until the end of a long chain, say, algorithm \( Q \). In the end we need some tool/algorithm to tune this \( Q \), which again comes back to the original question: How can we tune an algorithm \( A \) so that it can perform best?

It is worth pointing out that even if we have good tools to tune an algorithm, the best parameter setting and thus best performance all depend on the performance measures used in the tuning. Ideally, these parameters should be robust enough for minor parameter changes, random seeds, and even problem instance. However, in practice, they might not be achievable. According to Eiben [2], parameter tuning can be divided into iterative and noniterative tuners, single-stage and multistage tuners. The meanings of these terms are self-explanatory. In terms of the actual tuning, existing methods include sampling methods, screening methods, model-based methods, and metaheuristic methods. Their success and effectiveness can vary, and thus there are no well-established methods for universal parameter tuning.
12.3 Framework for Self-Tuning Algorithms

12.3.1 Hyperoptimization

From our earlier observations and discussions, it is clear that parameter tuning is the process of optimizing the optimization algorithm; therefore, it is a hyperoptimization problem. In essence, a tuner is a meta-optimization tool for tuning algorithms.

For a standard unconstrained optimization problem, the aim is to find the global minimum $f_*$ of a function $f(x)$ in a $d$-dimensional space. That is,

$$\min f(x), \quad x = (x_1, x_2, \ldots, x_d).$$

(12.4)

Once we choose an algorithm $A$ to solve this optimization problem, the algorithm will find a minimum solution $f_{\text{min}}$ that may be close to the true global minimum $f_*$. For a given tolerance $\delta$, this may require $t_{\delta}$ iterations to achieve $|f_{\text{min}} - f_*| \leq \delta$. Obviously, the actual $t_{\delta}$ will largely depend on both the problem objective $f(x)$ and the parameters $p$ of the algorithm used.

The main aim of algorithm tuning is to find the best parameter setting $p_*$ so that the computational cost or the number of iterations $t_{\delta}$ is the minimum. Thus, parameter tuning as a hyperoptimization problem can be written as

$$\min t_{\delta} = A(f(x), p),$$

(12.5)

whose optimality is $p_*$. Ideally, the parameter vector $p_*$ should be sufficiently robust. For different types of problems, any slight variation in $p_*$ should not much affect the performance of $A$, which means that $p_*$ should lie in a flat range rather than at a sharp peak in the parameter landscape.

12.3.2 A Multi-Objective View

If we look at the algorithm tuning process from a different perspective, it is possible to construct it as a multi-objective optimization problem with two objectives: one objective $f(x)$ for the problem $\Phi$ and one objective $t_{\delta}$ for the algorithm. That is,

$$\min f(x) \text{ and } \min t_{\delta} = A(f(x), p),$$

(12.6)

where $t_{\delta}$ is the (average) number of iterations needed to achieve a given tolerance $\delta$ so that the found minimum $f_{\text{min}}$ is close enough to the true global minimum $f_*$, satisfying $|f_{\text{min}} - f_*| \leq \delta$.

This means that for a given tolerance $\delta$, there will be a set of best-parameter settings with a minimum $t_{\delta}$. As a result, the bi-objectives will form a Pareto front. In principle, this bi-objective optimization problem (12.6) can be solved by any methods that are suitable for multi-objective optimization. But because $\delta$ is usually given, a natural way to solve this problem is to use the so-called $\epsilon$-constraint or $\delta$-constraint methods. The naming may be dependent on the notations; however, we will use $\delta$-constraints.
For a given $\delta \geq 0$, we change one of the objectives (i.e., $f(x)$) into a constraint, and thus the problem (12.6) becomes a single-objective optimization problem with a constraint. That is,

\[
\text{Minimize } t_\delta = A(f(x), p), \\
\text{subject to } f(x) \leq \delta.
\]

In the rest of this chapter, we set $\delta = 10^{-5}$.

The important thing is that we still need an algorithm to solve this optimization problem. However, the main difference from a common single objective problem is that the present problem contains an algorithm $A$. Ideally, an algorithm should be independent of the problem, which treats the objective to be solved as a black box. Thus we have $A(., p, \varepsilon)$. However, in reality, an algorithm will be used to solve a particular problem $\Phi$ with an objective $f(x)$. Therefore, both notations $A(., p)$ and $A(f(x), p)$ are used here.

### 12.3.3 Self-Tuning Framework

This framework was proposed by Yang et al. in 2013 [8]. In principle, we can solve (12.7) by any efficient or well-tuned algorithm. Now a natural question is: Can we solve this algorithm-tuning problem by the algorithm $A$ itself? There is no reason why we cannot. In fact, if we solve (12.7) using $A$, we have a self-tuning algorithm. That is, the algorithm automatically tunes itself for a given problem objective to be optimized. This essentially provides a framework for a self-tuning algorithm, as shown in Figure 12.1.

This framework is generic in the sense that any algorithm can be tuned this way and any problem can be solved within this framework. This essentially achieves two goals simultaneously: parameter tuning and optimality finding.

In the rest of this chapter, we use the firefly algorithm (FA) as a case study to self-tune FA for a set of function optimization problems.

---

**Implement an algorithm $A(., p, \varepsilon)$**

- with parameters $p = [p_1, ..., p_K]$ and random vector $\varepsilon = [\varepsilon_1, ..., \varepsilon_m]$;
- Define a tolerance (e.g., $\delta = 10^{-5}$);
- Algorithm objective $t_\delta(f(x), p, \varepsilon)$;
- Problem objective function $f(x)$;
- Find the optimality solution $f_{\min}$ within $\delta$;
- Output the number of iterations $t_\delta$ needed to find $f_{\min}$;
- Solve $\min t_\delta(f(x), p)$ using $A(., p, \varepsilon)$ to get the best parameters;
- Output the tuned algorithm with the best parameter setting $p^*_\ast$.

**Figure 12.1** A framework for a self-tuning algorithm.
12.4 A Self-Tuning Firefly Algorithm

Now let us use the framework outlined earlier to tune the firefly algorithm (FA). As we saw in the chapter on FA, it has the following updating equation:

\[ x_{i}^{t+1} = x_{i}^{t} + \beta_{0} e^{-\gamma r_{ij}^{2}} (x_{j}^{t} - x_{i}^{t}) + \alpha \epsilon_{i}^{t}, \] (12.9)

which contains four parameters: \( \alpha, \beta_{0}, \gamma, \) and the population size \( n. \) For simplicity of parameter tuning, we set \( \beta_{0} = 1 \) and \( n = 20, \) and therefore the two parameters to be tuned are \( \gamma > 0 \) and \( \alpha > 0. \) It is worth pointing out that \( \gamma \) controls the scaling, whereas \( \alpha \) controls the randomness. For this algorithm to converge properly, randomness should be gradually reduced, and one way to achieve such randomness reduction is to use

\[ \alpha = \alpha_{0} \theta^{t}, \quad \theta \in (0, 1), \] (12.10)

where \( t \) is the index of iterations/generations. Here \( \alpha_{0} \) is the initial randomness factor, and we can set \( \alpha_{0} = 1 \) without losing generality. Therefore, the two parameters to be tuned become \( \gamma \) and \( \theta. \)

Now we can tune FA for a set of five test functions. These functions can be found in the literature (e.g., [4]).

The Ackley function can be written as

\[ f_{1}(x) = -20 \exp \left[ -\frac{1}{5} \sqrt{\frac{1}{d} \sum_{i=1}^{d} x_{i}^{2}} \right] \exp \left[ \frac{1}{d} \sum_{i=1}^{d} \cos (2\pi x_{i}) \right] + 20 + e, \] (12.11)

which has a global minimum \( f_{*} = 0 \) at \((0, 0, \ldots, 0).\)

The simplest of De Jong’s functions is the so-called sphere function

\[ f_{2}(x) = \sum_{i=1}^{d} x_{i}^{2}, \quad -5.12 \leq x_{i} \leq 5.12, \] (12.12)

whose global minimum is obviously \( f_{*} = 0 \) at \((0, 0, \ldots, 0).\) This function is unimodal and convex.

Yang’s forest function [4],

\[ f_{3}(x) = \left( \sum_{i=1}^{d} |x_{i}| \right) \exp \left[ -\sum_{i=1}^{d} \sin (x_{i}^{2}) \right], \quad -2\pi \leq x_{i} \leq 2\pi, \] (12.13)

is highly multimodal and has a global minimum \( f_{*} = 0 \) at \((0, 0, \ldots, 0).\)

Rastrigin’s function,

\[ f_{4}(x) = 10d + \sum_{i=1}^{d} \left[ x_{i}^{2} - 10 \cos (2\pi x_{i}) \right], \quad -5.12 \leq x_{i} \leq 5.12, \] (12.14)

has a global minimum of \( f_{*} = 0 \) at \((0, 0, \ldots, 0).\) This function is highly multimodal.
Table 12.1 Results of parameter tuning for the firefly algorithm.

<table>
<thead>
<tr>
<th>Function</th>
<th>Mean $t_5 \pm \sigma_t$</th>
<th>Mean $\gamma \pm \sigma_{\gamma}$</th>
<th>Mean $\theta \pm \sigma_{\theta}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>589.7 ± 182.1</td>
<td>0.5344 ± 0.2926</td>
<td>0.9561 ± 0.0076</td>
</tr>
<tr>
<td>$f_2$</td>
<td>514.4 ± 178.5</td>
<td>0.5985 ± 0.2554</td>
<td>0.9540 ± 0.0072</td>
</tr>
<tr>
<td>$f_3$</td>
<td>958.1 ± 339.0</td>
<td>1.0229 ± 0.5762</td>
<td>0.9749 ± 0.0047</td>
</tr>
<tr>
<td>$f_4$</td>
<td>724.1 ± 217.6</td>
<td>0.4684 ± 0.3064</td>
<td>0.9652 ± 0.0065</td>
</tr>
<tr>
<td>$f_5$</td>
<td>957.2 ± 563.6</td>
<td>0.8933 ± 0.4251</td>
<td>0.9742 ± 0.0052</td>
</tr>
</tbody>
</table>

Zakharov’s function of

$$f_5(x) = \sum_{i=1}^{d} x_i^2 + \left( \frac{1}{2} \sum_{i=1}^{d} i x_i \right)^2 + \left( \frac{1}{2} \sum_{i=1}^{d} i x_i \right)^4$$

(12.15)

has a global minimum $f_5^* = 0$ at $(0, 0, \ldots, 0)$.

For each objective function, we run the FA to tune itself 50 times so as to calculate meaningful statistics. The population size $n = 20$ is used for all the runs. The means and standard deviations are summarized in Table 12.1, where $d = 8$ is used for all functions.

From this table, we can see that the variations of $\gamma$ are large, whereas $\theta$ has a narrow range. The best settings for parameters are problem dependent. These results imply the following:

- The optimal setting of parameters in an algorithm largely depends on the problem, and there is no unique best setting for all problems.
- The relatively large standard deviation of $\gamma$ means that the actual setting of $\gamma$ is not important to a given problem, and therefore there is no need to fine-tune $\gamma$. That is to say, a typical value of $\gamma = 1$ should work for most problems.
- Some parameters are more sensitive than others. In the present case, $\theta$ needs more fine-tuning due to its smaller standard deviations.

These findings confirm the earlier observations in the literature that $\gamma = O(1)$ can be used for most applications [5–7], whereas $\alpha$ needs to reduce gradually in terms of $\theta$. That is probably why other forms of probability distributions such as Lévy flights may lead to better performance than the random numbers drawn from the Gaussian normal distribution.

12.5 Some Remarks

Parameter tuning is the process of tuning an algorithm to find the best parameter settings so that an algorithm can perform the best for a given set of problems. However, such parameter tuning is a very tough optimization problem. In fact, such hyperoptimization is the optimization of an optimization algorithm, which requires special care because the optimality depends on both the algorithm to be tuned and the problem to be solved. It is
possible to view this parameter-tuning process as a bi-objective optimization problem; however, the objectives involve an algorithm, and thus this bi-objective problem is different from the multi-objective problem in the normal sense.

Our framework for self-tuning algorithms is truly self-tuning in the sense that the algorithm to be tuned is used to tune itself. We have used the firefly algorithm and a set of test functions to test the proposed self-tuning algorithm framework. Results have shown that it can indeed work well. We also found that some parameters require fine-tuning, but others do not need to be tuned carefully. This is because different parameters may have different sensitivities and thus may affect the performance of an algorithm in different ways. Only parameters with high sensitivities need careful tuning.

Though successful, the present framework requires further extensive testing with a variety of test functions and many different algorithms. It may also be possible to see how probability distributions can affect the tuned parameters and even the parameter-tuning process. In addition, it can be expected that this present framework is also useful for parameter control, so a more generalized framework for both parameter tuning and control can be used for a wide range of applications. Furthermore, our current framework may be extended to multi-objective problems so that algorithms for multi-objective optimization can be tuned in a similar way.

References


