6th ICML Workshop on Automated Machine Learning (2019)

Accelerating the Nelder–Mead Method with Predictive Parallel Evaluation

Yoshihiko Ozaki OZAKI-Y@AIST.GO.JP
AI Research Center, AIST & GREE, Inc.

Shuhei Watanabe SHUHEI.WATANABE@AIST.GO.JP
AI Research Center, AIST & The University of Tokyo

Masaki Onishi ONISHI@NI.AIST.GO.JP
AI Research Center, AIST

Abstract

The Nelder–Mead (NM) method has been recently proposed for application in hyperparameter optimization (HPO) of deep neural networks. However, the NM method is not suitable for parallelization, which is a serious drawback for its practical application in HPO. In this study, we propose a novel approach to accelerate the NM method with respect to the parallel computing resources. The numerical results indicate that the proposed method is significantly faster and more efficient when compared with the previous naive approaches with respect to the HPO tabular benchmarks.

1. Introduction

Hyperparameter optimization (HPO), which involves the automatic configuration of the hyperparameters of machine learning models, is currently one of the most important problems in automated machine learning (AutoML) research (Feurer and Hutter, 2019; Hutter et al., 2019). Various methods, such as Bayesian optimization (Hutter et al., 2011; Bergstra et al., 2011; Snoek et al., 2012), have been studied for solving this problem over the past decades.

It has been recently reported that the Nelder–Mead (NM) method (Nelder and Mead, 1965) finds good configurations while the method converges with fewer objective function evaluations than Bayesian optimization or the covariance matrix adaption evolution strategy (CMA-ES) (Loshchilov and Hutter, 2016) in the HPO of the convolutional neural networks (Ozaki et al., 2017). However, in the era of cloud computing, a large number of computing resources, such as tens or hundreds of Graphics Processing Units (GPUs), are easily available in parallel, and it is essential to effectively use these parallel computing resources.

Therefore, the methods that are not suitable for parallelization are less attractive options. Unlike batch Bayesian optimization or the CMA-ES, suitable for parallelization, each operation of the NM method has a fixed number of evaluations determined by the dimensionality of the search space. This signifies that only a limited number of evaluations can be naively parallelized in each iteration and that the speed of the method cannot be easily increased. This is a significant drawback of the NM method that should be overcome to ensure its practical applications. In this study, we propose a novel approach to accelerate the NM method with parallel computing resources.

2. Related Work

2.1 Hyperparameter Optimization

Machine learning models, such as deep neural networks, are generally very sensitive to their hyperparameters. However, it is difficult to manually obtain a proper configuration for maximizing the performance of the models. HPO is a research field that attempts to automatically configure the hyperparameters of machine learning models by optimization algorithms rather than manually. In HPO, the following optimization problem is solved:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in \mathbf{X}, \end{array} \tag{1}$$

where $\mathbf{X} = X_1 \times X_2 \times \cdots \times X_N$ is the N-dimensional configuration space of the hyperparameters and $f(x)$ is the performance metric (e.g., cross-validation loss) of the target model configured by $x \in \mathbf{X}$. By solving (1), the optimal configuration is determined as $x^\star = \operatorname{argmin}_x f(x)$.

The most common HPO algorithms are grid search and random search (Bergstra and Bengio, 2012). These algorithms are not particularly powerful but they are still extensively used because of their simplicity. The Bayesian optimization (Hutter et al., 2011; Bergstra et al., 2011; Snoek et al., 2012) and population-based methods (Loshchilov and Hutter, 2016; Lorenzo et al., 2017) are effective approaches. The former uses a surrogate to approximate the computationally expensive objective function and an acquisition function that controls the exploration-exploitation trade-off of the search. In contrast, the latter approach performs optimization by updating the population based on the fitness values of the individuals that can be evaluated in parallel.

Multi-fidelity is currently a growing trend in case of HPO research. Multi-fidelity methods drastically reduce the computation time using techniques such as training data subsampling (Klein et al., 2017a; Li et al., 2018) and learning curve prediction (Domhan et al., 2015; Klein et al., 2017b).

A number of other studies on HPO methods, including the NM method (Cohen et al., 2005; Ozaki et al., 2017) (Section 2.2), have been conducted. A recent survey (Feurer and Hutter, 2019) provides additional information on HPO.

2.2 Nelder–Mead Method

The NM method (see Algorithm 1) is a well-known derivative-free optimization heuristic. Over the previous fifty years, this method has been extensively used with great success in a variety of applications, including HPO. The NM method minimizes the objective function by iteratively applying operations, such as `order`, `reflect`, `expand`, `inside contract`, `outside contract`, and `shrink`, to vertices $S = \{x^{(0)}, \ldots, x^{(N)}\}$ whose convex hull, $\mathrm{conv}(S)$, is a non-degenerate simplex in the search space. This method performs a maximum of four sequential operations and evaluates at most $N + 2$ points per iteration. The evaluations in the method are sequential, with the exception of the evaluations in its initialization and the `shrink` operation, which can be trivially parallelized. The standard choice of coefficients is $\gamma^s = 0.5$, $\delta^{ic} = -0.5$, $\delta^{oc} = 0.5$, $\delta^r = 1$, and $\delta^e = 2$, and $\mathrm{diam}(\mathrm{conv}(S))$ is defined as $\max_{0 \le i < j \le N} \|x^{(i)} - x^{(j)}\|_2$. Graphical representations of the NM method are provided in Appendix A.

Algorithm 1 Nelder–Mead method

```
    $f : \mathbb{R}^N \to \mathbb{R}$                                          ▷ objective function
    $K \in \mathbb{Z}_{>0}$                                              ▷ maximum number of iterations
    $\epsilon \in \mathbb{R}_{>0}$                                              ▷ minimum diameter of simplex
    $0 < \gamma^s < 1, -1 < \delta^{ic} < 0 < \delta^{oc} < \delta^r < \delta^e$
    $S^{(0)} = \{x^{(n)} \in \mathbb{R}^N \mid n = 0, \dots, N\}$ s.t. $\mathrm{conv}(S^{(0)})$ is a $N$-simplex
 1: function Nelder–Mead
 2:     for $k = 0, 1, \dots$ do
 3:         $S^{(k)} = \{x^{(0)}, \dots, x^{(N)} \mid f(x^{(0)}) \le \dots \le f(x^{(N)})\}$          ▷ order
 4:         if $k == K$ or $\mathrm{diam}(\mathrm{conv}(S^{(k)})) \le \epsilon$ then
 5:             return $x^{(0)}$
 6:         $x^c = \frac{1}{N}\sum_{n=0}^{N-1} x^{(n)}$
 7:         $x^r = x^c + \delta^r(x^c - x^{(N)})$                              ▷ reflect
 8:         if $f(x^{(0)}) \le f(x^r) < f(x^{(N-1)})$ then $S^{(k+1)} = \{x^{(0)}, \dots, x^{(N-1)}, x^r\}$ continue
 9:         else if $f(x^r) < f(x^{(0)})$ then
10:             $x^e = x^c + \delta^e(x^c - x^{(N)})$                          ▷ expand
11:             if $f(x^e) \le f(x^r)$ then $S^{(k+1)} = \{x^{(0)}, \dots, x^{(N-1)}, x^e\}$ continue
12:             else $S^{(k+1)} = \{x^{(0)}, \dots, x^{(N-1)}, x^r\}$ continue
13:         else if $f(x^r) < f(x^{(N)})$ then
14:             $x^{oc} = x^c + \delta^{oc}(x^c - x^{(N)})$                    ▷ outside contract
15:             if $f(x^{oc}) \le f(x^r)$ then $S^{(k+1)} = \{x^{(0)}, \dots, x^{(N-1)}, x^{oc}\}$ continue
16:         else if $f(x^r) \ge f(x^{(N)})$ then
17:             $x^{ic} = x^c + \delta^{ic}(x^c - x^{(N)})$                    ▷ inside contract
18:             if $f(x^{ic}) < f(x^{(N)})$ then $S^{(k+1)} = \{x^{(0)}, \dots, x^{(N-1)}, x^{ic}\}$ continue
19:         $S^{(k+1)} = \{x^{(0)} + \gamma^s(x^{(n)} - x^{(0)}) \mid n = 0, \dots, N\}$         ▷ shrink
```

In the field of HPO research, Cohen et al. (2005) applied the NM method to optimize the hyperparameters of the support vector machines (SVMs). This method achieved comparable results with those achieved by grid search with approximately a quarter of the number of evaluations. More recently, Ozaki et al. (2017) used the NM method to optimize the hyperparameters of convolutional neural networks for performing image classification. The results revealed that the NM method found good configurations faster than random search, coordinate search, Bayesian optimization, and CMA-ES.

Several studies have examined the parallelization of the NM method. One approach involves parallel speculative evaluations of the objective function on the points generated by all the operations at each iteration (Dennis and Torczon, 1988; Mariano et al., 2013). This approach evaluates $N+4$ points (1 point each for `reflect`, `expand`, `inside contract`, and `outside contract`, and N points for `shrink`) per iteration. This method is simple and suitable for hardware implementation; however, it performs many evaluations that are likely to be unnecessary. Another approach involves additional evaluations performed in parallel on points that are not searched by the standard NM method (Coetzee and Botha, 1998; Lewis et al., 2006). This approach focuses on improving the search ability of the method rather than increasing its speed.

3. Proposed Method

3.1 Nelder–Mead Method with Predictive Parallel Evaluation

To increase the speed of the NM method, speculative evaluation of the points required after more than one iteration should be implemented using parallel computing resources. However, the number of candidate points increases exponentially per speculative iteration;

Algorithm 2 Monte Carlo simulation

```
   $f : \mathbb{R}^N \to \mathbb{R}$                    ▷ the objective function of Algorithm 1
   $\mathcal{D}$ : defined in Eq. (3)                   ▷ past observations of $f$
   $I \in \mathbb{Z}_{>0}$                              ▷ number of Monte Carlo samples
   $J \in \mathbb{Z}_{>0}$                              ▷ number of speculative iterations
   $L \in \mathbb{Z}_{>0}$                              ▷ the line number of Algorithm 1
   $k, K$                                               ▷ $k$ and $K$ of Algorithm 1
1: function MonteCarloSimulation
2:     function $g(x)$
3:         $\mu(x), \sigma^2(x)$ : Eq. (4), (5) with the past observations $\mathcal{D}$
4:         return a sample from $\mathcal{N}(\mu(x), \sigma^2(x))$
5:     for $i = 1, \ldots, I$ do
6:         replace $f$ with $g$ and run $\min(J, K - k)$ more iterations of the Algorithm 1 from the line $L$
7:     calculate $\mathcal{C} = \{(x, \text{count}(x)) \mid x \in \mathcal{X}\}$ s.t. $\mathcal{X}$ is the set of points evaluated in the simulation and count$(x)$ is
   the corresponding number of evaluations
8:     return $\mathcal{C}$
```

thus, the approach that evaluates all the candidates becomes impractical quickly as the number of speculative iterations increases. Thus, the candidates for speculative evaluation must be selected using a thoughtful approach.

The novel method that has been proposed in this study involves speculative evaluation by predicting the points that are likely to be evaluated in future iterations. We use a probabilistic surrogate model of the objective function and perform a Monte Carlo simulation on the surrogate to determine the points that have to be speculatively evaluated. A Gaussian process (GP) regression model (Williams and Rasmussen, 2006) is used as a surrogate.

In GP regression, we assume that our target function f follows a GP as follows:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')), \tag{2}$$

where $m(x)$ is the mean function and $k(x, x')$ is the covariance function. In this study, we consider $m(x) = 0$ and the Matérn kernel for $k(x, x')$. Further, the predictive mean and standard deviation of the target function can be calculated based on the past M observations, the set of pairs of a point, and the corresponding function value

$$\mathcal{D} = \{(x^{(1)}, f(x^{(1)})), \ldots, (x^{(M)}, f(x^{(M)}))\}, \tag{3}$$

as follows:

$$\mu(x^{(M+1)}) = \mathbf{k}^\top (\mathbf{K} + \sigma^2_{\text{noise}} \mathbf{I})^{-1} \mathbf{f}, \tag{4}$$
$$\sigma^2(x^{(M+1)}) = k(x^{(M+1)}, x^{(M+1)}) - \mathbf{k}^\top (\mathbf{K} + \sigma^2_{\text{noise}} \mathbf{I})^{-1} \mathbf{k}, \tag{5}$$

where $\sigma^2_{\text{noise}}\mathbf{I}$ is the additive noise, and

$$\mathbf{f} = \begin{bmatrix} f(x^{(1)}) & \cdots & f(x^{(M)}) \end{bmatrix}^\top, \quad \mathbf{k} = \begin{bmatrix} k(x^{(M+1)}, x^{(1)}) & \cdots & k(x^{(M+1)}, x^{(M)}) \end{bmatrix}^\top, \quad \mathbf{K} = \begin{bmatrix} k(x^{(1)}, x^{(1)}) & \cdots & k(x^{(1)}, x^{(M)}) \\ \vdots & \ddots & \vdots \\ k(x^{(M)}, x^{(1)}) & \cdots & k(x^{(M)}, x^{(M)}) \end{bmatrix}. \tag{6–8}$$

The ranking of candidates is determined by running a Monte Carlo simulation, as outlined in Algorithm 2. The x with the maximum count(x) in $\mathcal{C}$ is most likely to be evaluated in future iterations and should be speculatively evaluated at first.

Algorithm 3 describes the NM method with predictive parallel evaluation. It should be noted that fewer than P candidates may be generated in lines 5–6.

Algorithm 3 Nelder–Mead method with predictive parallel evaluation

```
   $P \in \mathbb{Z}_{>0}$                                  ▷ number of parallel computing resources
1: function Nelder–MeadMethodwithPredictiveParallelEvaluation
2:     initialize Algorithm 1 and evaluate $\{f(x) \mid x \in S^{(0)}\}$ in parallel
3:     while the condition of line 4 of Algorithm 1 is not met do
4:         run Algorithm 1 until reaching an unevaluated point
5:         calculate $\mathcal{C}$ by running Algorithm 2 from the line that an unevaluated point has appeared in Algorithm 1
6:         evaluate the top-$P$ unevaluated points with high count$(x)$ value in $\mathcal{C}$ in parallel
7:     return the result of Algorithm 1 (or the $x$ which has the minimum $f(x)$ in the past observations $\mathcal{D}$)
```

3.2 Practical Considerations

In HPO, the computational cost of the simulation using the proposed method is generally significantly lower than that of the objective function evaluation. Regardless, after a large number of evaluations, the GP is computationally expensive because it requires a time of $O(M^3)$. In this case, creating a surrogate with only relatively recent observations makes it possible to reduce the computational cost with only a small impact on the prediction performance.

To avoid the evaluation of the candidate points generated by the simulation that are not likely to be required in future iterations, the minimum value for count(x) can be set as the decision threshold for speculation. However, evaluating these points is useful for improving the accuracy of a surrogate. In addition, it is possible that one of these points exhibits the minimum value even though it may not be the convergent point of the NM method. Therefore, our recommendation is to perform speculative evaluations with the largest number of parallel computing resources at any given time.

4. Numerical Results

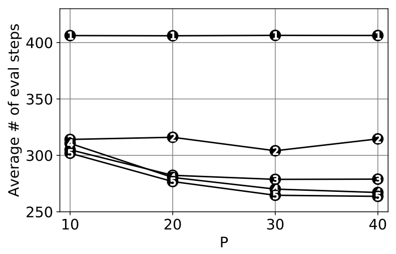
To test the efficiency of our proposed approach, we performed empirical tests with respect to the three tabular benchmarks for HPO (FCNetProteinStructureBenchmark, FCNetSliceLocalizationBenchmark, and FCNetYearPredictionBenchmark) (Klein et al., 2018) [1]. In the benchmarks, six numerical hyperparameters were optimized, whereas the remaining hyperparameters, which were categorical, were set to the best-known configurations. For the points in the search space for which there was no data in the table, the objective function values were calculated by linear interpolation. For points outside the search space, the values were set to a large constant value, 10^9. In the NM method, discrete parameters were considered to be continuous, and rounding was applied before performing the evaluation.

First, we compared the proposed method [2] with the two baseline methods in terms of the number of sequential evaluation steps (the evaluations in parallel were counted as one evaluation step) and the number of evaluations with a fixed number of parallel computing resources $P = N + 4 = 10$. The baseline method 1 performed fully parallel evaluations in its initialization and `shrink` operation but did not perform speculative evaluation. The baseline method 2 performed speculative evaluations of $N + 4$ candidate points generated by all the operations per iteration. We further compared the proposed method for different numbers of parallel computing resources ($P = 10$, 20, 30, and 40). In both the experiments,

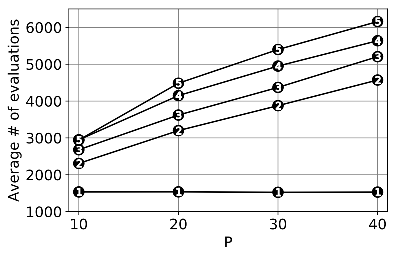
1. `https://github.com/automl/nas_benchmarks`
2. The implementation is available at `https://github.com/y0z/NMwithPredictiveParallelEvaluation`.

Method	J	Average # of eval steps	Average # of evaluations
Baseline 1	-	590.27 ($\pm$141.42)	614.10 ($\pm$142.82)
Baseline 2	-	347.27 ($\pm$89.32)	3469.67 ($\pm$893.21)
Proposed	1	406.20 ($\pm$97.24)	1534.20 ($\pm$427.69)
	2	314.13 ($\pm$72.26)	2307.83 ($\pm$558.02)
	3	304.97 ($\pm$54.57)	2679.13 ($\pm$464.80)
	4	310.60 ($\pm$67.58)	2948.20 ($\pm$642.62)
	5	301.90 ($\pm$58.70)	2942.33 ($\pm$567.27)

Table 1: Performance comparison between the proposed method and baseline methods



(a) Average # of eval steps v.s. P



(b) Average # of evaluations v.s. P

Figure 1: Performance vs. P for $J = 1, \ldots, 5$

we set $\epsilon = 10^{-4}$, $K = 500$, and $I = 100$ and used only the most recent 100 observations to create a surrogate. Optimization was performed 10 times for each benchmark with different initial simplices. The experimental results are presented in Table 1 and Figure 1.

Table 1 demonstrates that our proposed method involved significantly fewer evaluation steps than those involved in the baseline methods 1 and 2. Our method was up to approximately 49% faster than the baseline method 1 and up to approximately 13% faster than the baseline method 2. In terms of the number of evaluations, our proposed method used hundreds of fewer evaluations than that required by the baseline method 2. In summary, our proposed method is the fastest and performs speculative function evaluations more efficiently than that performed using baseline method 2. Figure 1 indicates that the number of evaluation steps can be reduced by increasing P and J. However, $\frac{\text{decrease in the number of eval steps}}{\text{increase in the number of evaluations}}$ decreases for large P and J values. This can be observed because the success rate of the speculative evaluations decreases as J increases.

5. Conclusion

In this study, we proposed the NM method with predictive parallel evaluation. We tested our method with respect to the HPO benchmarks, and the numerical results indicated that our method is much faster and more efficient when compared with the naive parallelization methods. The proposed method may be useful not only in case of HPO but also for other AutoML problems, such as neural architecture search, and considerably general blackbox optimization problems. We have several unverified ideas that may improve our method such as surrogates other than GP. Additional tests will be conducted in a future work.

References

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

Louis Coetzee and Elizabeth C Botha. The parallel downhill simplex algorithm for unconstrained optimisation. *Concurrency: Practice and Experience*, 10(2):121–137, 1998.

Gilles Cohen, Patrick Ruch, and Mélanie Hilario. Model Selection for Support Vector Classifiers via Direct Simplex Search. In *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference, Clearwater Beach, Florida, USA*, pages 431–435, 2005.

JE Dennis and Virginia Torczon. Parallel Implementations Of The Nelder-Mead Simplex Algorithm For Unconstrained Optimization. In *High Speed Computing*, volume 880, pages 187–192. International Society for Optics and Photonics, 1988.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Matthias Feurer and Frank Hutter. Hyperparameter Optimization. In *Automated Machine Learning: Methods, Systems, Challenges* Hutter et al. (2019), pages 3–33.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer International Publishing, 2019.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *Artificial Intelligence and Statistics*, pages 528–536, 2017a.

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning Curve Prediction with Bayesian Neural Networks. In *International Conference on Learning Representations (ICLR) 2017 Conference Track*, 2017b.

Aaron Klein, Eric Christiansen, Kevin Murphy, and Frank Hutter. Towards reproducible neural architecture and hyperparameter search. In *ICML 2018 Workshop on Reproducibility in ML (RML 2018)*, 2018.

Andrew Lewis, David Abramson, and Tom Peachey. RSCS: A parallel simplex algorithm for the Nimrod/O optimization toolset. *Scientific Programming*, 14(1):1–11, 2006.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sanchez Ramos, and José Ranilla Pastor. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 481–488. ACM, 2017.

Ilya Loshchilov and Frank Hutter. CMA-ES for Hyperparameter Optimization of Deep Neural Networks. In *International Conference on Learning Representations (ICLR) 2016 Workshop Track*, 2016.

Artur Mariano, Paulo Garcia, and Tiago Gomes. SW and HW speculative Nelder-Mead execution for high performance unconstrained optimization. In *2013 International Symposium on System on Chip (SoC)*, pages 1–5. IEEE, 2013.

John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

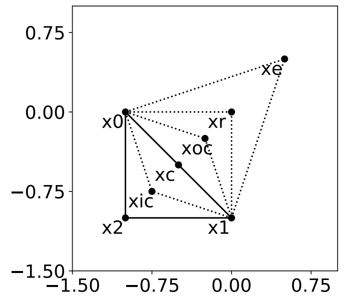
Yoshihiko Ozaki, Masaki Yano, and Masaki Onishi. Effective hyperparameter optimization using Nelder-Mead method in deep learning. *IPSJ Transactions on Computer Vision and Applications*, 1(9):1–12, 2017.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

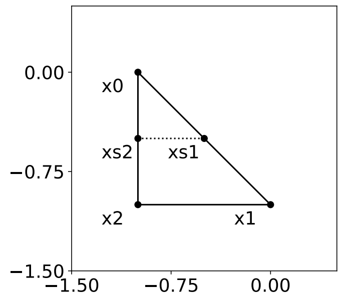
Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. MIT Press Cambridge, MA, 2006.

Appendix A. Graphical Representations of the Nelder–Mead Method

Graphical representations are quite useful to understand the NM method. Sample executions of the operations of the NM method are presented in Figure 2. A sample execution of the NM method is presented in Figure 3.

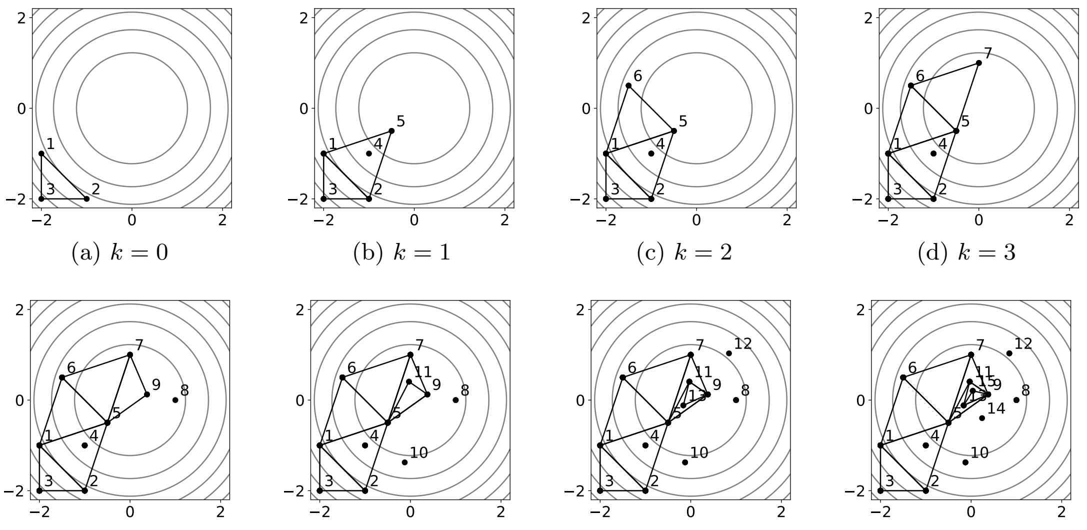


(a) Example: `reflect` (xr), `expand` (xe), `inside contract` (xic) and `outside contract` (xoc)



(b) Example: `shrink` (xs1 and xs2)

Figure 2: The NM method operation examples



(a) $k = 0$ (b) $k = 1$ (c) $k = 2$ (d) $k = 3$

(e) $k = 4$ (f) $k = 5$ (g) $k = 6$ (h) $k = 7$

Figure 3: Optimization of $f(x, y) = x^2 + y^2$ using the NM method (numbers indicate the order of evaluations)