Asynchronous Classification-Based Optimization*

Yu-Ren Liu, Yi-Qi Hu, Hong Qian and Yang Yu
Natoinal Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China
{liuyr, huyq, qianh, yuy}@lamda.nju.edu.cn

Abstract

Asynchronous parallelization is an effective way to accelerate optimization. While asynchronous parallelization can destroy the sequential structure of optimization algorithms, it has been found counter-intuitively that some optimization algorithms are proven to preserve their performance under asynchronous parallelization, including the stochastic gradient descent for first-order optimization of differentiable functions and Pareto optimization for zeroth-order optimization in binary space. Following this direction, in this paper, we show that the classification-based optimization, which is a recently developed framework for zeroth-order optimization in continuous space, can also enjoy the asynchronous parallelization. We implement ASRACOS, an asynchronous version of a classification-based optimization algorithm SRACOS, to accelerate the optimization through asynchronous parallelization. Experiments on synthetic functions and controlling tasks in OpenAI Gym demonstrate that ASRACOS can achieve almost linear speedup while preserving good solution quality.

1 Introduction

Asynchronous parallelism is an effective way to accelerate optimization. However, for sequential update optimization algorithms, asynchronous parallelism can destroy the sequential structure of the optimization, which deteriorates the optimization performance. It has been found that some optimization algorithms are proven to preserve their performance under asynchronous parallelization, including the stochastic gradient decent for first-order optimization of differentiable functions [21] and Pareto optimization for zeroth-order optimization in binary space [14].

In this paper, we focus on derivative-free optimization, which regards the objective function f as a black-box function: given a solution x, only the function value $f(x)$ is available. Other information of f such as the gradient is unavailable. Derivative-free optimization methods can be roughly categorized into three classes: model-based methods, deterministic Lipschitz optimization methods and meta-heuristic search. Model-based methods, such as Bayesian optimization methods [2, 5, 15], learn a model from the solutions and the model is then applied to guide sampling of solutions for the next round. Deterministic Lipschitz optimization methods need Lipschitz continuity assumption on f, such as [1, 9, 11, 12]. Meta-heuristic search is designed with inspired heuristics, such as evolutionary strategies [6, 7, 10, 13].

Classification-based optimization is a recently developed theoretical framework of model-based derivative-free optimization methods, where the model is implemented by a classification model discriminating good solutions from bad ones. Its implementation, the SRACOS algorithm [8], has shown outstanding performance in various applications [18–20]. However, the sequential structure of SRACOS keeps it from being parallized, which is unbearable for time-consuming optimization tasks.

In this paper, we propose an asynchronous classification-based optimization algorithm ASRACOS. Our theoretical analyses and empirical studies verify the effectiveness of the proposed method. In particular, we make the following key contributions:

- We apply a feasible modification to SRACOS to make it parallelizable, and implement its asynchronous version ASRACOS, which holds the sequential structure while being able to utilize multiple servers.
- We provide the $(\epsilon - \delta)$ query complexity bound of ASRACOS in theoretical analyses and further give the condition when ASRACOS can achieve a better (worse) performance than SRACOS using the same number of evaluations.
- We empirically compare ASRACOS with several other parallel classification-based optimization algorithms on four synthetic testing functions, and apply them to direct policy search for 6 controlling tasks, where an artificial neural network is used as the policy and optimized. Experiment results show that ASRACOS can achieve almost linear speedup while preserving good solution quality.

The rest four sections present the background, the proposed ASRACOS algorithm, the empirical results, and the conclusion, respectively.

2 Background

Derivative-free optimization, also termed as zeroth-order or black-box optimization, involves a class of optimization algorithms that does not rely on gradient information. We consider general minimization problems in continuous domains. Let X denote a bounded solution space that is a

*This work is supported by NSFC (61876077), Jiangsu SF (BK20170013), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

The 1st International Conference on Distributed Artificial Intelligence (DAI 2019), Beijing, China.

Algorithm 1 Sequential RACOS (SRACOS)

Input:

f: Objective function to be minimized;
C: A binary classification algorithm;
λ: Balancing parameter;
$m \in \mathbb{N}^+$: Number of negative samples;
$k \in \mathbb{N}^+ (\leq m)$: Number of positive samples;
$r = m + k$;
$N \in \mathbb{N}^+$: Budget, i.e., number of evaluations;
`Sampling`: Sampling sub-procedure;
`Selection`: Decide the positive/negative solutions;
`Replace`: Replacing sub-procedures.

Procedure:

1: Collect $S = \{x_1, ..., x_r\}$ by i.i.d. sampling from $\mathcal{U}_X$
2: $B = \{(x_1, y_1), ..., (x_r, y_r)\}, \forall x_i \in S : y_i = f(x_i)$
3: $(B^+, B^-) =$ `Selection`$(B; k)$
4: Let $(\tilde{x}, \tilde{y})) = \text{argmin}_{(x,y)\in B^+} y$
5: **for** $t = r + 1$ *to* N **do**
6: $\quad h = C(B^+, B^-)$
7: $\quad x = \begin{cases} \texttt{Sampling}(\mathcal{U}_{D_h}) & \text{w.p. } \lambda \\ \texttt{Sampling}(\mathcal{U}_X) & \text{w.p. } 1 - \lambda \end{cases}$
8: $\quad y = f(x)$
9: $\quad [(x', y'), B^+] =$ `Replace`$((x, y), B^+,$ 'strategy_P')
10: $\quad [, B^-] =$ `Replace`$((x', y'), B^-,$ 'strategy_N')
11: $\quad (\tilde{x}, \tilde{y}) = \text{argmin}_{(x,y)\in B^+ \cup \{(\tilde{x},\tilde{y})\}} y$
12: **end for**
13: **return** $(\tilde{x}, \tilde{y})$

compact subset of $\mathbb{R}^n$, and $f : X \to \mathbb{R}$ denote a minimization problem. Assume that there exists x^* such that $f(x^*) = \min_{x\in X} f(x)$. Let F denote the set of all functions that satisfy this assumption. Given $f \in F$, the minimization problem is to find a solution $x^* \in X$ s.t. $\forall x \in X : f(x^*) \leq f(x)$. For black-box optimization, given a solution x, only the objective function $f(x)$ is accessible for evaluating x.

Model-based derivative-free optimization algorithms share a framework that iteratively learns a model for promising search areas and samples solutions from the model. Different kinds of methods usually vary in the design of the model. For example, cross-entropy methods [4] may use Gaussian distribution as the model, Bayesian optimization methods [15] employ Gaussian process to model the joint distribution, and the estimation of distribution algorithms have incorporated many kinds of learning models. Classification-based optimization algorithms learn a particular type of model: classification model. A classification model learns to classify solutions into two categories, *good* or *bad*. Then solutions are sampled from the *good* areas.

SRACOS[8] is a recently proposed classification-based optimization algorithm. Unlike other model-based optimization algorithms, the sampling region of SRACOS is learned by a simple classifier, which maintains an axis-parallel rectangle to cover all the positive but no negative solutions. SRACOS shows outstanding performance both in theoretical analyses and empirical studies. Its pseudo-code is presented in Algorithm 1. To initialize, SRACOA samples a batch of solutions. We will get a solution-value tuple set B after querying objective function for each solution in S (line 1 to 2). After that, Selection sub-procedure is used to split B into two tuple sets B^+ and B^-, where the positive set B^+ is consisted of the best k solutions and the negative set B^- is consisted of the rest. (line 3). Line 4 and line 11 record the best-so-far solution-value tuple. In the following loop, SRACOS trains a binary classifier C to learn an axis-parallel region, which contains a randomly selected positive solution in B^+ and rules out all the negative solutions in B (line 6). More details about the classifier C can be found in [17]. Then, a new solution is uniformly sampled from this learned region with probability λ or uniformly sampled from the whole solution space with probability $1 - \lambda$ (line 7). After evaluating the new sampled solution (line 8, the most time-consuming part), the solution-value tuple is used to update B^+, B^- and the best-so-far solution-value tuple accordingly (line 9 to 11). `Replace`$(a, A,$'s') subprocedure replaces a tuple in the set A with a according to a strategy 's'. There are three strategies proposed in [8]: replacing the worst solution in A (WR-), randomly replacing a solution in A (RR-), and replacing the solution in A which has the largest margin from the best-so-far solution (LM-). Note that 'strategy_P' can only be 'WR-', and 'strategy_N' can be any one of these three strategies. Finally, SRACOS will return the best-so-far tuple $(\tilde{x}, \tilde{y})$ (line 13).

3 Asynchronous SRACOS

The idea of making SRACOS parallelizable is straightforward: Sample N_s (the number of evaluation servers) solutions, rather than 1 solution, after initialization. Then these solutions can be evaluated parallelly. Whenever an evaluation is finished, the method will update the model and sample the next solution for evaluation. Note that the sequential update structure is still held through the modification.

With the same as Algorithm 1, `Selection`$(B; k)$ splits the solution-value tuple set B into a positive set and a negative set, where the positive set contains k best-so-far tuples. `Replace`$(a, A,$'s') means replacing a tuple from the set A with a according to some strategy 's', and the replaced tuple and the updated tuple set will be returned. $\lambda - \texttt{Sampling}_n(\mathcal{U}_{D_h}, \mathcal{U}_X)$ means sampling n solutions and for each solution, it is sampled from the distribution $\mathcal{U}_{D_h}$ with probability λ and $\mathcal{U}_X$ with probability $1 - \lambda$.

The proposed Asynchronous SRACOS is shown in Algorithm 2. After initialization, ASRACOS will get two tuple sets B^+ and B^- according to function values (line 3). Then a binary classifier is trained on the basis of these two sets to learn the potential high-quality region in the solution space (line 4). The learned region contains one selected good solution in the positive set and rules out all the bad solutions in the negative set. ASRACOS contains two first-in-first-out blocking queues: D for the unevaluated solutions and E for the evaluated solutions. D and E are also set to be shared between the main thread and evaluation threads for data communication. D is initialized with the first batch of

Algorithm 2 Asynchronous SRacos (ASRacos)

Input: (extra input than SRacos)
$N_s \in \mathbb{N}^+$: The number of evaluation servers;
Procedure:
1: Collect $S = \{x_1, ..., x_r\}$ by i.i.d. sampling from $\mathcal{U}_X$
2: $B = \{(x_1, y_1), ..., (x_r, y_r)\}, \forall x_i \in S : y_i = f(x_i)$
3: $(B^+, B^-) =$ `Selection`$(B; k)$
4: $h_1 = C(B^+, B^-)$
5: $D, E =$ *SharedQueue*$\{\}$, *SharedQueue*$\{\}$
6: $D = \{x_{r+1}, ..., x_{r+N_s}\} = \lambda -$ `Sampling`$_n(\mathcal{U}_{D_h}, \mathcal{U}_X)$
7: Run `Evaluation(D, E)` sub-procedures on N_s daemon threads
8: **for** $t = r + 1$ *to* N **do**
9: $(x, y) =$ `take`(E)
10: $[(x', y'), B^+] =$ `Replace`$((x, y), B^+,$ 'strategy_P')
11: $[, B^-] =$ `Replace`$((x', y'), B^-,$ 'strategy_N')
12: $(\tilde{x}, \tilde{y}) = \text{argmin}_{(x,y) \in B^+ \cup \{(\tilde{x}, \tilde{y})\}} y$
13: $h = C(B^+, B^-)$
14: $x = \lambda -$ `Sampling`$_1(\mathcal{U}_{D_h}, \mathcal{U}_X)$
15: `put`(x, D)
16: **end for**
17: **return** $(\tilde{x}, \tilde{y})$
18:
19: `Evaluation(D, E):`
20: **while** true **do**
21: $x =$ `take`(D)
22: $y = f(x)$.
23: `put`$((x, y), E)$
24: **end while**

sampled solutions and E is initialized to be empty (line 5 and 6). Then, ASRacos starts N_s evaluation servers (implemented in the newly created threads), each keeping evaluating a solution taken from D and putting the result (x, y) into E (line 21 to 23). In the following loop, ASRacos takes the evaluated tuple (x, y) from E and uses it to update the tuple set B^+ and B^- (line 9 to 11). Once a new binary classifier C is trained (line 13), a new solution will be sampled and put into D (line 14, 15).

In summary, ASRacos divides the sequential evaluation and update procedure in SRacos into two components: the asynchronous evaluation component and the sequential model update component. The asynchronous evaluation component can make use of multiple servers, while the model update component can still update the classification model sequentially, which holds the sequential structure in SRacos. The blocking queue D and E are created for data communication between threads.

Figure 1 demonstrates the flow charts of the optimization procedure of ASRacos and SRacos, where the solid arrow denotes the sampling and evaluation procedure, the hollow arrow denotes an update on the data distribution Dt and si, sj and sk denote the unused solutions sampled before. It can be observed that Dt is always updated by the solution sampled from Dt for SRacos, while it can be updated by the solution sampled from another ditribution several iterations ago for ASRacos, which causes the difference of the data distribution of two algorithms. The next section discusses the effect of such difference on the query complexity of ASRacos.

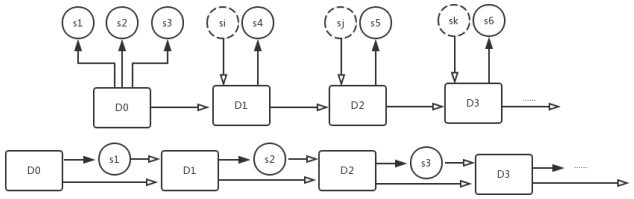


Figure 1: The flow charts of the optimization procedure of ASRacos (up, using 3 servers) and SRacos (down).

4 Theoretical analysis

For a subset $D \subseteq X$, let $\#D = \int_{x \in X} \mathbb{I}[x \in D] dx$, where $I[\cdot]$ is the indicator function. Define $|D| = \#D/\#X$. Let $D_\alpha = \{x \in X | f(x) \leq \alpha\}$, and $D_\epsilon = \{x \in X | f(x) - f(x^*) < \epsilon\}$ for $\epsilon > 0$. A hypothesis is a mapping $h : X \to \{1, +1\}$. Let $H \subseteq \{h : X \to \{1, +1\}\}$ be a hypothesis space. Let $D_h = \{x \in X | h(x) = +1\}$ for hypothesis $h \in H$, i.e., the positive class region represented by h. Denote $\mathcal{U}_{D_h}$ the uniform distribution over D_h and τ_h the distribution defined on D_h induced by h respectively. Let $S_t = \lambda \mathcal{U}_{D_{h_t}} + (1 - \lambda)\mathcal{U}_X$ be the sampling distribution in iteration t, D_t be the distribution under which the classifier is trained in iteration t, R_{D_t} denote the generalization error of $h_t \in H$ under the distribution D_t, D_{KL} denote the Kullback-Leibler (KL) divergence between two probability distributions and N denote the number of iterations. The superscript S is added to the symbols to represent SRacos and A is added to represent ASRacos

The complexity of an algorithm is measured by the (ϵ, δ)-query complexity as Definition 1 [16, 17]. It counts the total number of calls to the objective function by an algorithm before it finds a solution that reaches the approximation level ϵ, with high probability.

Definition 1 *((ϵ, δ)-Query Complexity)*
Given $f \in F$, an algorithm A, $0 < \delta < 1$ and $\epsilon > 0$, the (ϵ, δ)-query complexity is the number of calls to f such that, with probability at least $1 - \delta$, A finds at least one solution $\tilde{x} \in X \subseteq \mathbb{R}^n$ satisfying $f(\tilde{x}) - f(x^*) \leq \epsilon$, where $f(x^*) = min_{x \in X} f(x)$.

Then we derive a upper bound of the query complexity of ASRacos under the conditions of error-target θ-dependence and γ-shrinking rate [17].

Lemma 1 *Given $f \in \mathcal{F}$, $0 < \delta < 1$ and $\epsilon > 0$, if* ASRacos *has error-target θ-dependence and γ-shrinking rate, then its (ϵ, δ)-query complexity is upper bounded by*

$$O(\max\{\frac{1}{|D_\epsilon|}((1-\lambda)+\frac{\lambda}{\gamma(N-r)}\sum_{t=r+1}^{N}\Phi_t^A)^{-1}ln\frac{1}{\delta},N\})$$

where $\Phi_t^A = (1 - R_{D_t^A} - \#X\sqrt{\frac{1}{2}D_{KL}(D_t^A||\mathcal{U}_X)} - \theta) \cdot |D_{\alpha_t}|^{-1}$ *and* $\#X$ *is the volume of* X.

We omit the proof of Lemma 1 because the proof is the same as the proof of Theorem 1 in [8] except for the value of R_{D_t} and $D_{KL}(D_t||\mathcal{U}_X)$ at each iteration. By Lemma 1, we can have a comparison of the query complexity bound of ASRACOS and the bound of SRACOS. The result is shown in Theorem 1.

Theorem 1 *Ignoring the constant factor and fixing* θ *and* γ*, ASRacos can have a better (or worse) query complexity upper bound than SRacos if for any iteration* t*:*

$$R_{D_t^A} - R_{D_t^S} < (>)\#X(\sqrt{\frac{1}{2}D_{KL}(D_t^S||\mathcal{U}_X)} - \sqrt{\frac{1}{2}D_{KL}(D_t^A||\mathcal{U}_X)})$$

Theorem 1 discloses that if the difference of the training distribution between two algorithms is greater than the difference of generalization error, ASRACOS can be better than SRACOS even if using the same number of evaluations. Moreover, ASRACOS can use nearly N_s times more evaluations compared to SRACOS within the same time. So it is much easier for ASRACOS to find a better solution than SRACOS in actual use. The proof of Theorem 1 is presented in the appendix due to the space limitation.

5 Experiments

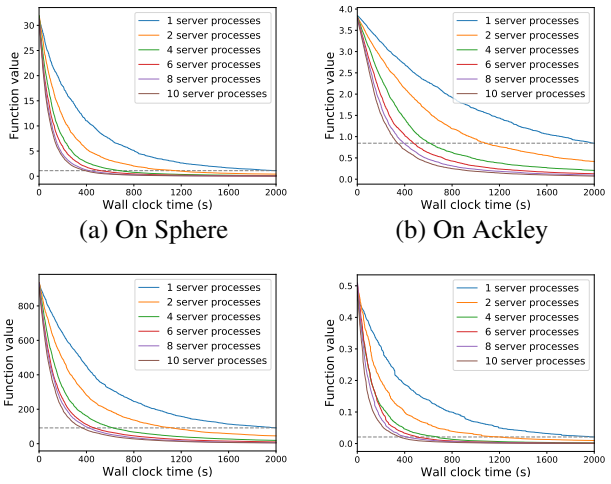
We evaluate the performance of ASRACOS in two environments. One is the optimization of classical synthetic functions, containing a convex function and three highly non-convex functions; the other is the controlling tasks in OpenAI Gym, an open source environment for reinforcement learning research.

We investigate the properties of the asynchronous parallelism on classification-based optimization methods, including convergence rate, speedup and solution quality. We compare our method with another two parallel classification-based methods: Parallel RACOS (PRACOS) and Parallel SRACOS (PSRACOS). PRACOS is a simple parallel implementation of the batch-mode method RACOS [17]. PSRACOS shares the same structure with ASRACOS, and only varies in that the classification model will not update until the slowest evaluation server finishes evaluation. Note that when the number of evaluation servers is 1, ASRACOS and PSRACOS are equivalent to SRACOS, and PRACOS equals RACOS. [8] has compared the performance of a sequential classification-based optimization algorithm with other state-of-the-art derivative-free optimization algorithms, so we omit these comparisons in this paper.

5.1 On Synthetic Functions

We choose four benchmark testing functions: the convex Sphere function and the highly non-convex Ackley, Rastrigin and Griewank function. They are defined as: $Sphere(x) = \sum_{i=1}^{d} x_i^2$, $Ackley(x) = -20e^{-\frac{1}{5}\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}} - e^{\frac{1}{d}\sum_{i=1}^{d}cos(2\pi x_i)} + 20 + e$, $Rastrigin(x) = 10d + \sum_{i=1}^{d}[x_i{}^2 - 10\ cos(2\pi x_i)]$ and $Griewank(x) = \sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_{i=1}^{d}cos(\frac{x_i}{\sqrt{i}}) + 1$. Graphs of these functions are presented in the appendix.

All of the functions are minimized within the solution space $X = [-1, 1]^d$, of which the minimum value is 0 and the optimal solution is $(0, 0, ..., 0)$. During implementation, we choose $d = 100$ and shift the optimal solution by 0.2, which means the new optimal solution is $(0.2, 0.2, ..., 0.2)$, to avoid possible optimization bias to the origin point. In addition, we add a fixed 1-second sleep for each evaluation. This is a reasonable modification since any distributed algorithm faces the networking overhead. If the evaluation time cost is even smaller than the networking overhead, parallelization may not be necessary. Another 1-second sleep with 0.25 probability is also added to simulate a situation where evaluation servers vary in computational performance, i.e. some servers are explicitly slower than others, which is common in real-world applications. Each algorithm is repeated 10 times independently, and the average performance is reported.

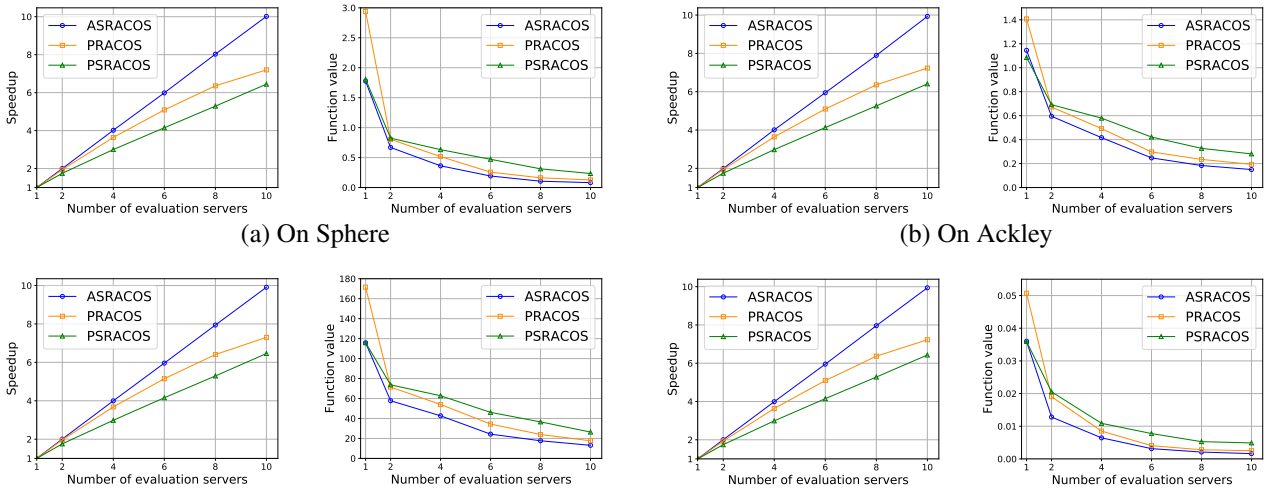


(a) On Sphere (b) On Ackley

(c) On Rastrigin (d) On Griewank

Figure 2: Comparison of the convergence rate with the number of evaluation servers $N_s = 1, 2, 4, 6, 8, 10$.

On convergence rate. We firstly study the convergence rate of ASRACOS. We set the time for optimization to be 2000 seconds and compare the performance with the number of evaluation servers $N_s = 1, 2, 4, 6, 8, 10$. The results are shown in Figure 2. The dotted line represents the optimal value that ASRACOS obtains when uisng one server



(a) On Sphere (b) On Ackley

(c) On Rastrigin (d) On Griewank

Figure 3: On each objective function, left: speedup, right: the average of the function value (the one closer to 0 the better).

(also the result of SRACOS). It can be observed that ASRACOS with more evaluation servers reduces the objective function value with a higher rate, indicating that the asynchronous parallelism can accelerate the convergence.

On speedup. We then study the speedup w.r.t the number of evaluation servers (N_s). We set the budget to be 2000 for each algorithm and calculate the speedup as $S_i = \frac{T_1}{T_i}$, where T_i represents the time consumed when $N_s = i$. The results are shown in Figure 3. From the left plots of each function, we can observe that ASRACOS (blue line) achieves linear speedup, notably better than PRACOS and PSRACOS. The results reflect the advantage of asynchronous parallelism over simple parallelism when servers vary in computational performance.

On solution quality. To study the solution quality w.r.t. the number of evaluation servers within the same time constraint, we set the time for optimization to be 20 minutes for each algorithm. The results are shown in the right plots of Figure 3. We can see that algorithms using more servers get better solution quality and ASRACOS achieves the best performance among them.

5.2 On Controlling Tasks in OpenAI Gym

OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. The toolkit provides many controlling tasts, from which we choose 'Acrobot', 'MountainCar', 'Pendulum', 'HalfCheetah', 'Swimmer' and 'Ant' to investigate the speedup and solution quality of ASRACOS.

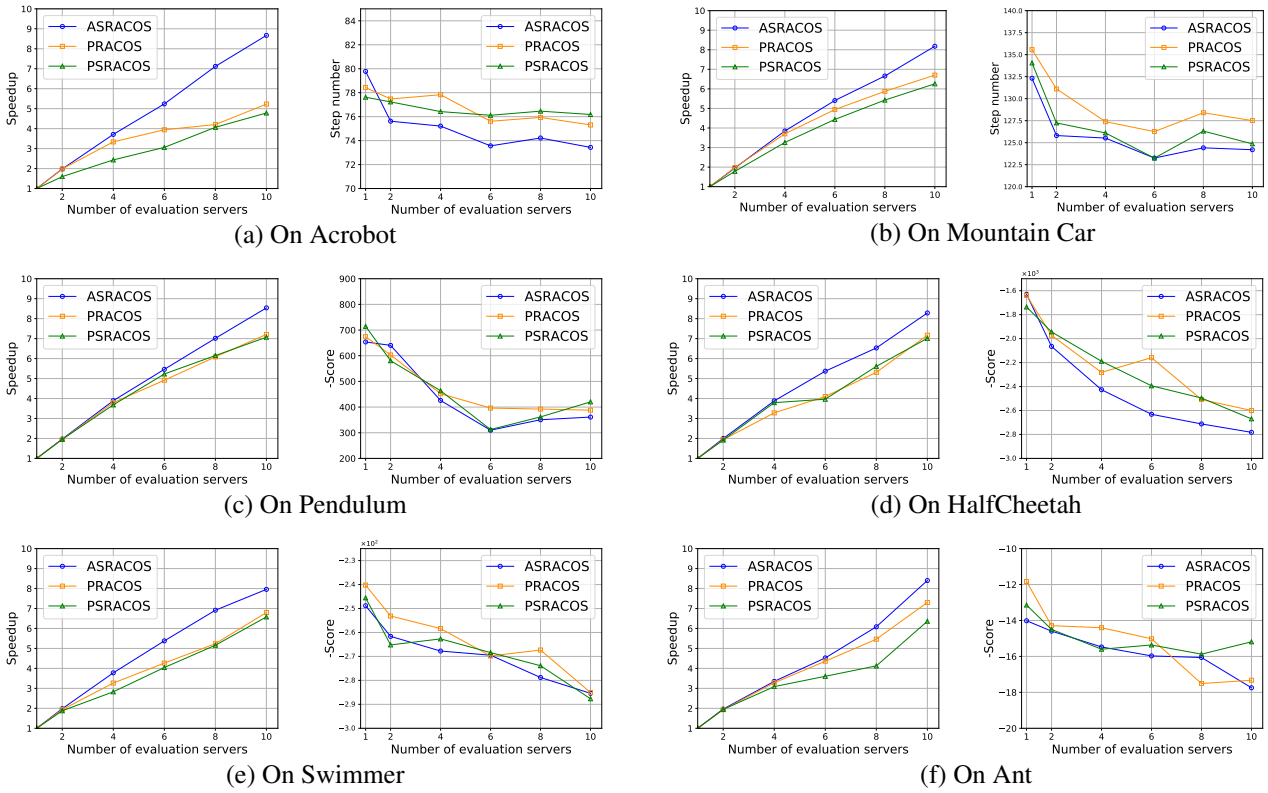
We use the framework of direct policy search to solve these tasks. Direct policy search employs optimization algorithms to search in the parameter space of a policy for maximizing the cumulative reward. The policy is often represented by a neural network [3], whose weights

Table 1: Parameters of the Gym tasks

Task	d_{State}	#Actions	NN nodes	#Weights	Horizon
Acrobot-v1	6	1	5, 3	48	500
MountainCar-v0	2	1	5	15	200
Pendulum-v0	3	1	5	20	200
HalfCheetah-v2	17	6	10	230	1000
Swimmer-v2	8	2	5, 3	61	1000
Ant-v2	111	8	15	1785	1000

$w = \{w_1, w_2, ..., w_n\}$ are the parameters to be optimized. The neural network takes the observation of the state as input and outputs an action according to its policy. After that, it will get the reward of that action and the observation of the next state. This interaction can be repeated until the game is over or the maximum step is reached. The cumulative reward is used as an evaluation of the policy network, i.e. $f(w)_i = \sum_{t=1}^{T} R_t$. The agent would have different cumulative rewards if the initial state is reset to be different, so we take the avarage of multiple simulations as the final evaluation value of one neural network: $f(w) = \sum_{i=1}^{m} f(w)_i/m$, which can reduce the noise to some extent. In a nutshell, our aim is to find the optimal parameter w for this network so as to achieve the best performance. We list the task information and the settings of neural network in Table 1, where d_{State}, #Actions, NN nodes, #Weights and Horizon respectively denote the dimension size of observation, the dimension size of action, the hidden layers of the neural network, the total number of parameters in the neural network and the maximum step.

Details of each task can be found in the homepage of OpenAI Gym (http://gym.openai.com). Among these tasks, 'Acrobot' and 'MountainCar' are finding policies with the smallest step number to achieve the goal. Other tasks are to find policies to get score from the environment as high as possible. The average cumulative reward of 200 simu-



(a) On Acrobot (b) On Mountain Car

(c) On Pendulum (d) On HalfCheetah

(e) On Swimmer (f) On Ant

Figure 4: For each task, left: speedup, right: the mean step (Acrobot, MountainCar) or −Score (Pendulum, HalfCheetah, Swimmer, Ant) of the best found policy (the smaller y-axis coordinate value the better).

lations is used as the evaluation value of one network for 'Acrobot', 'MountainCar' and 'Pendulum. And for other tasks, the average reward of 20 simulations is used. The solution space X is set to be $[-10, 10]^{\#\text{Weight}}$. The output of the neural network is scaled to be within the action space, which is defined by the environment. Each algorithm is repeated 10 times and the mean value of the top-5 results is reported. The results are plotted in Figure 4.

On speedup. We set the budget to be 2000 for each algorithm. From the left plots of each task, we can observe that ASRACOS (blue line) can still achieve almost linear speedup, better than PRACOS and PSRACOS. Due to the inevitable competition for computing resource, the speedup ratio in these environments is smaller than that on synthetic functions, which merely simulate the time-consuming tasks simply by adding sleep operations. In addition, for 'Acrobot', 'MountainCar' and 'Ant', a better solution would make the game stop earlier, which consumes less evaluation time, and result in a lower speedup.

On solution value. We convert the maximization problems in 'Pendulum', 'HalfCheetah', 'Swimmer' and 'Ant' to the minimization problems by adding a minus to the score. The time for optimization is set to be 20 minutes for each algorithm. From the right plots in each subfigure, we can see that the algorithm using more serves can get better solution quality in most cases. Nevertheless, in some cases, the algorithm may get worse solution quality. The reason is that in one case there exists randomness in the process of optimization, in another the evaluation is inaccurate under noisy environments, which may make a bad solution seem to be good and lead the optimization to the wrong direction. Similar to the results of the synthetic functions, ASRACOS achieves the best performance in most cases.

6 Conclusion

In this paper, we propose an asynchronous classification-based optimization method, ASRACOS, for accelerating the optimization. We analyze the query complexity of ASRACOS, and further provide the condition when ASRACOS can achieve a better (worse) performance than SRACOS using the same number of evaluations. Experiments on synthetic functions show that ASRACOS can achieve higher convergence rate when having more evaluation servers. On both synthetic functions and direct policy search for controlling tasks, ASRACOS demonstrates almost linear speedup and gets a better solution quality than other parallel algorithms. Future work includes combining noise-handling methods into ASRACOS to speed up the optimization in noisy environments, and applying ASRACOS to large-scale optimization problems in real world.

References

[1] Sébastien Bubeck, Gilles Stoltz, and Jia Yuan Yu. Lipschitz bandits without the Lipschitz constant. In *Proceedings of the 22nd International Conference on Algorithmic Learning Theory*, pages 144–158, Espoo, Finland, 2011.

[2] Adam D. Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12:2879–2904, 2011.

[3] Andres El-Fakdi Marc Carreras and Narcís Palomeras. Direct policy search reinforcement learning for robot control. In *Artificial Intelligence Research and Development, Proceedings of the 8th International Conference of the ACIA*, pages 9–16, Alguer, Italy, 2005.

[4] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, 2005.

[5] Nando de Freitas, Alexander J. Smola, and Masrour Zoghi. Exponential regret bounds for gaussian process bandits with deterministic observations. In *Proceedings of the 29th International Conference on Machine Learning*, Edinburgh, UK, 2012.

[6] David E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, 1989.

[7] Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[8] Yi-Qi Hu, Hong Qian, and Yang Yu. Sequential classification-based optimization for direct policy search. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 2029–2035, San Francisco, CA, 2017.

[9] Donald R. Jones, Cary D. Perttunen, and Bruce E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.

[10] Pedro Larrañaga and Jose A. Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science; Business Media, 2001.

[11] Rémi Munos. From bandits to Monte-Carlo Tree Search: The optimistic principle applied to optimization and planning. *Foundations and Trends in Machine Learning*, 7(1):1–130, 2014.

[12] Arnold Neumaier. Global optimization in action, continuous and Lipschitz optimization: Algorithms, implementations and applications. nonconvex optimization and its applications. *Journal of Global Optimization*, 12(3):319–321, 1998.

[13] Frank Neumann and Ingo Wegener. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378(1):32–40, 2007.

[14] Chao Qian, Jing-Cheng Shi, Yang Yu, Ke Tang, and Zhi-Hua Zhou. Parallel Pareto optimization for subset selection. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 1939–1945, New York, NY, 2016.

[15] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pages 2960–2968, Lake Tahoe, NV, 2012.

[16] Yang Yu and Hong Qian. The sampling-and-learning framework: A statistical view of evolutionary algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 149–158, Beijing, China, 2014.

[17] Yang Yu, Hong Qian, and Yi-Qi Hu. Derivative-free optimization via classification. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 2286–2292, Phoenix, AZ, 2016.

[18] Yang Yu, Wei-Yang Qu, Nan Li, and Zimin Guo. Open category classification by adversarial sample generation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3357–3363, Melbourne, Australia, 2017.

[19] Jianbing Zhang, Yixin Sun, Shujian Huang, Cam-Tu Nguyen, Xiaoliang Wang, Xinyu Dai, Jiajun Chen, and Yang Yu. AGRA: An analysis-generation-ranking framework for automatic abbreviation from paper titles. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 4221–4227, Melbourne, Australia, 2017.

[20] Wen-Ji Zhou, Yang Yu, and Min-Ling Zhang. Binary linear compression for multi-label classification. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3546–3552, Melbourne, Australia, 2017.

[21] Martin Zinkevich, Markus Weimer, Alexander J. Smola, and Lihong Li. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23*, pages 2595–2603, British Columbia, Canada, 2010.

A Appendix

A.1 Proof of Theorem 2

In order to explicitly compare the query complexity of ASRacos with that of SRacos, we let $\mathcal{D}_t^A$ and $\mathcal{D}_t^S$ denote the distribution under which the classifier is trained in iteration t of ASRacos and SRacos, and $R_{\mathcal{D}_t^A}$ and $R_{\mathcal{D}_t^S}$ denote the generalization error of them, respectively.

Lemma 1. *Given $f \in \mathcal{F}$, $0 < \delta < 1$ and $\epsilon > 0$, if* ASRacos *has error-target θ-dependence and γ-shrinking rate, then its (ϵ, δ)-query complexity is upper bounded by*

$$O(\max\{\frac{1}{|D_\epsilon|}((1-\lambda) + \frac{\lambda}{\gamma(N-r)}\sum_{t=r+1}^{N}\Phi_t^A)^{-1} ln\frac{1}{\delta}, N\})$$

where $\Phi_t^A = (1 - R_{D_t^A} - \#X\sqrt{\frac{1}{2}D_{KL}(D_t^A||\mathcal{U}_X)} - \theta) \cdot |D_{\alpha_t}|^{-1}$ *and $\#X$ is the volume of X.*

Theorem 1. *Ignoring the constant factor and fixing θ and γ, ASRacos can have a better (or worse) query complexity upper bound than SRacos if for any iteration t:*

$$R_{D_t^A} - R_{D_t^S} < (>) \#X(\sqrt{\frac{1}{2}D_{KL}(D_t^S||\mathcal{U}_X)} - \sqrt{\frac{1}{2}D_{KL}(D_t^A||\mathcal{U}_X)})$$

Before proving Theorem 1, we first recall the (ϵ,δ)-query complexity bound of a classification-based sequential derivative-free optimization algorithm which has been derived in [8].

Theorem 2 ([8]) *Given $f \in \mathcal{F}$, $0 < \delta < 1$ and $\epsilon > 0$, if a classification-based sequential derivative-free optimization algorithm has error-target θ-dependence and γ-shrinking rate, then its (ϵ,δ)-query complexity is upper bounded by*

$$O\left(\max\left\{\frac{1}{|D_\epsilon|}\left((1-\lambda) + \frac{\lambda}{\gamma(N-r)}\sum_{t=r+1}^{N}\Phi_t^S\right)^{-1}\ln\frac{1}{\delta}, N\right\}\right),$$

where $\Phi_t^S = \left(1 - R_{D_t^S} - \#X\sqrt{\frac{1}{2}D_{KL}(D_t^S||\mathcal{U}_X)} - \theta\right)\cdot|D_{\alpha_t}|^{-1}$ *and $\#X$ is the volume of X.*

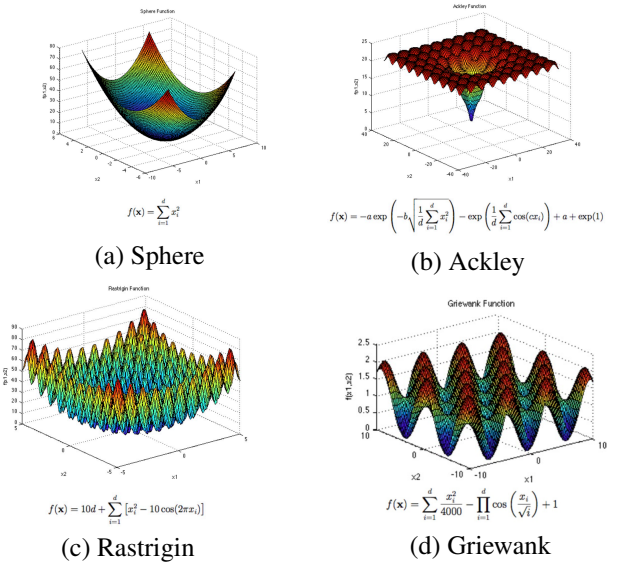
Proof of Theorem 2

Proof. In Lemma 1, ignoring the constant factor and letting $\epsilon > 0$ be small enough such that we only need to focus on the part of

$$\frac{1}{|D_\epsilon|}\left((1-\lambda) + \frac{\lambda}{\gamma(N-r)}\sum_{t=r+1}^{N}\Phi_t^A\right)^{-1}\ln\frac{1}{\delta},$$

where $\Phi_t^A = \left(1 - R_{D_t^A} - \#X\sqrt{\frac{1}{2}D_{KL}(D_t^A||\mathcal{U}_X)} - \theta\right)\cdot|D_{\alpha_t}|^{-1}$ and $\#X$ is the volume of X. On the basis of Lemma 1 and Theorem 2, to compare ASRacos with SRacos, it is sufficient to compare the part of $1 - R_{D_t^A} - \#X\sqrt{\frac{1}{2}D_{KL}(D_t^A||\mathcal{U}_X)} - \theta$ with $1 - R_{D_t^S} - \#X\sqrt{\frac{1}{2}D_{KL}(D_t^S||\mathcal{U}_X)} - \theta$ if we ignore the corresponding constant factors. It can be verified directly that, for any iteration t, if $R_{D_t^A} - R_{D_t^S} < \#X(\sqrt{\frac{1}{2}D_{KL}(D_t^S||\mathcal{U}_X)} - \sqrt{\frac{1}{2}D_{KL}(D_t^A||\mathcal{U}_X)})$, then ASRacos is better than SRacos; if $R_{D_t^A} - R_{D_t^S} > \#X(\sqrt{\frac{1}{2}D_{KL}(D_t^S||\mathcal{U}_X)} - \sqrt{\frac{1}{2}D_{KL}(D_t^A||\mathcal{U}_X)})$, then ASRacos is worse than SRacos. ■

A.2 Synthetic Functions



(a) Sphere (b) Ackley

(c) Rastrigin (d) Griewank

Figure 5: Graphs of four synthetic functions (http://www.sfu.ca/ ssurjano/optimization.html).