A Study of Asynchronous Budgeted Optimization

R. Le Riche[1,2] **R. Girdziušas**[1] **J. Janusevskis**[3,1]
[1] École Nationale Supérieure des Mines de Saint-Étienne, [2] CNRS UMR 6158, France
[3] Riga Technical University, Latvia

Abstract

Budgeted optimization algorithms based on Gaussian processes have attracted a lot of attention as a way to deal with computationally expensive cost functions. Recently, parallel versions of these algorithms have further improved their ability to address the computation cost bottleneck. This article focuses on those algorithms that maximize the multi-point expected improvement criterion. Synchronous and asynchronous parallel versions of such algorithms are compared. It is shown that asynchronous algorithms are slower than the synchronous ones iteration-wise. In terms of wall clock time however and contrarily to synchronous algorithms, asynchronous implementations benefit from a near-linear speed-up up to the order of 100 computing nodes. This speed-up is bounded by the maximization of the expected improvement time which, by becoming a critical time limitation, may change the way researchers look at budgeted optimization methods.

1 Introduction

Budgeted optimization methods deal with a limited number of cost function evaluations, which is often less than one thousand [1]. Our research is motivated by such practical demand stemming from the car industry where the objective function involves a three-dimensional turbulent fluid flow simulation [2] so that any sequential optimization approach takes days to complete. In an attempt to substantially reduce the optimization time, it was decided to take advantage of the now available distributed computing infrastructures (here the ProActive PACA Grid cloud [3])

Many parallelized budgeted optimization algorithms, see e.g. [4, 5, 6, 7, 8], rely on a sequential Gaussian process updating technique, first introduced by M. Schonlau [9], which creates a set (a "generation") of candidate solutions by maximizing one-point acquisition functions. These methods are attractive because they build a global Gaussian model of the data, which makes them adapted to costly objective functions, and allows to tune most of their parameters from the data. Some approaches [10, 11], including our work, maximize the multi-point acquisition function directly. However, all of these methods are limited to small generation sizes, often less than ten, when a typical distributed computing infrastructure offers over a hundred nodes.

In this work, we first point out in Section 2 that the scalibility in terms of number of computing nodes used and the accompanying total execution time of the parallel optimization algorithm vastly depend on the way the function evaluations are distributed. Then in Section 3, we empirically compare the performance of multi-point expected improvement (EI) optimization algorithms. Versions of the algorithm differ in *i)* the node access mode (synchronous vs. asynchronous) and *ii)* the definition of the multi-point EI which can include or not a memory of actively running simulations.

2 Synchronous versus Asynchronous Node Access

We consider the optimization of an expensive-to-evaluate function defined over the domain $\Omega \subseteq \mathbb{R}^d$. Typically, $d < 10$, and Ω will be the hyper-box made from the known upper and lower bounds of the optimization variables. Let $m = \mu + \lambda$ be the number of nodes, i.e. the number of virtual machines (computers) available on a remote cloud to evaluate "expensive functions", and suppose that the access to the cloud is possible every time λ nodes provide a result. Typically, $\lambda \ll m$, such as $\lambda = 1, 2, 3, 4$, while $m = 32, 100$. Let a generic optimization algorithm generate sets (generations) of λ candidate solutions whose objective functions need to be evaluated.

The simplest case is the *synchronous node access* when a generation of solutions is sent for evaluation to the cloud, and calculation of new candidates resumes only when the whole generation is evaluated. The drawback here is that the time between the submission of each consecutive generation of points, which we will refer to as *the wall clock time* (t_{WC}), is determined by the slowest cost function evaluation. In the *asynchronous node access*, one first fills all the m available nodes with initial random solutions, and then each node is updated immediately as it becomes available. Here “immediately” implies a time scale of minutes when the function evaluations take hours to complete. The asynchronous node access has been studied previously [11, 1], but with different acquisitions and too little attention paid to the real time characteristics.

We now introduce a simple node access timing model. Let T be the set of $m = \mu + \lambda$ elements t_i, each element being the time of one expensive function evaluation at the ith remote node. The t_i’s are initially taken as independent uniform random variables between t_{min} and t_{max}. We write t_b the blocking time which is the time it takes to calculate and send λ new points to update the free nodes. The model consists of a simple recipe to calculate the wall clock time:

$s = 1$ (λ-generation counter) *i)* Find the λ smallest elements of T (not necessarily distinct), and create the set S out of them $S = \{t_{1:m}, t_{2:m}, \ldots, t_{\lambda:m}\}$, *ii)* $t_{\lambda:m}$ is the largest element in S, compute the wall clock time, i.e., the update node time $t_{WC} = t_b + t_{\lambda:m}$, *iii)* Create the set $M = T \setminus S$, and replace every element t of M with $\max(0, t - t_{WC})$, finally, *iv)* Update the set T with uniform numbers between t_{min} and t_{max}, $T = M \cup \mathcal{U}[t_{\text{min}}, t_{\text{max}}]^{\lambda}$ *v)* $s \leftarrow s + 1$, if $s \leq s_{\text{max}}$ goto *i)*.

Fig. 1a represents the average wall clock time, $E(T_{WC}) \pm$ std. dev., as a function of the number of nodes. The average is performed over 100 runs and for $s \leq s_{\max} = 250$. It is seen that, irrespectively of the optimizer that generates the λ candidate solutions, the asynchronous node access has $E(T_{WC}) \approx t_b + E(T_{\lambda:m})/m$. It allows the wall clock time to decrease with the number of nodes m as $\mathcal{O}(m^{-1})$. However, the time the optimizer needs to generate the λ candidate solutions, t_b, is a lower bound (hence the blocking time name) which is rapidly reached. The time of asynchronous optimizers should be compared to that of their synchronous pendant which is on the average $t_b + E(t_{\lambda:\lambda}) = t_b + \frac{\lambda}{\lambda+1}(t_{\text{max}} - t_{\text{min}}) + t_{\text{min}}$ and tends for large λ to $t_b + \mathcal{O}(t_{\text{max}})$. Both synchronous and asynchronous wall clock times illustrate Amdhal’s law [12] where t_b is the non parallelized part of the code. We now refine this analysis for a specific optimizer based on multi-point expected improvement. Specifically, we study the effect of asynchronicity and λ on its efficiency. We shall demonstrate that the decrease in wall clock time dominates the increase in number of function evaluations needed to reach a target value.

3 Synchronous and asynchronous multi-point EI optimizers

We now work with the family of multi-point EI optimization algorithms that have been described in [4, 10, 13]. An initial design of experiments (DOE) made of p points is created with the random Latin Hyper-Cube Sampling method. These points are then sent to the cloud for evaluation of their (expensive) objective functions. Once λ nodes are free while μ points, $\mathbf{x}_{1\ldots\mu}$, are being actively evaluated, the algorithm finds the next λ points by maximizing a joint expected improvement criterion:

$$\mathbf{x}_s^* = \arg \max_{\mathbf{x}_s \in \mathrm{R}^{d\lambda}} \mathrm{E}\left(\max\left(0, \left(f_{\min} - \min\left(Y(\mathbf{x}_{1\ldots\mu}), Y(\mathbf{x}_s)\right)\right)\right) | A_{s-1} \right), \tag{1}$$

where A_s denotes the known ($\mathbf{x}$, $Y(\mathbf{x})$) values gathered after s λ-generations, $f_{\min}$ is the currently observed best solution, $Y(\mathbf{x}_{1\ldots\mu}) = (Y(\mathbf{x}_1), \ldots, Y(\mathbf{x}_\mu))$ and $Y(\mathbf{x}_s) = (Y(\mathbf{x}_{\mu+1}), \ldots, Y(\mathbf{x}_{\mu+\lambda}))$ are points of a Gaussian process conditioned to interpolate the known values A_{s-1} (kriging model). The expectation will be abbreviated as $\mathrm{EI}^{\mu,\lambda}$. Maximizing $\mathrm{EI}^{\mu,\lambda}$ is attractive for our purpose because it is a natural, parameter free, traduction of what one might want to do to create a parallel global search: maximize an expected progress over a set of points while accounting for all known observations and points that are currently being evaluated. $\mathrm{EI}^{\mu,\lambda}$ is estimated by Monte Carlo simulations with 1000 samples. The maximization on $\mathbf{x}_s$ is performed by the CMA-ES [14] algorithm with a population size of 10 points and 500 iterations.

The μ points correspond to the candidate locations whose expensive function values are being actively evaluated, but are not known at the time when a request comes to send a new candidate for the evaluation. Including active points

$x_{1\dots\mu}$ in the target part of the EI criterion prevents the algorithm from resampling there because $\mathrm{EI}^{\mu,\lambda}$ decreases as some of the new λ search points get closer to active points [10, 13].

4 Tests and discussion

Table 1: Test functions

Label	Cost function	Domain	Minimal value	Modality
"michalewicz2d"	$\sum_{i=1}^{2} \sin(x_i)\sin^2(ix_i^2/\pi)$	$[0,5]^2$	-1.841	multimodal
"rosenbrock6d"	$\sum_{i=1}^{5} 100(x_{i+1}-x_i^2)^2+(1-x_i)^2$	$[0,5]^6$	0	unimodal
"rank1approx9d"	$\Vert\mathbf{A}_{4\times5}-\mathbf{x}_{1\dots4}\mathbf{x}_{5\dots9}^T\Vert_2$, $a_{ij}\sim U(0,1)$[1]	$[-1,1]^9$	0.712	bimodal

The performance of the algorithms is assessed by repeating 100 minimizations of the three artificial functions described in Table 1. The optimization quality is measured through the normalized real improvement (NRI) defined as

$$\mathrm{NRI}(s)=\frac{f_0-f_{\text{min}}(s)}{f_0-f_{\text{true}}}. \tag{2}$$

Here f_0 is the smallest value of the cost function achieved on the initial DOE, f_{min} denotes the value achieved after a particular generation of points is evaluated, and f_{true} is the true ideal minimal value, which is given in Table 1. In the context of parallel calculation, it is necessary to consider two performance measures: the speed-up per generation S_G, and the real time speed-up S_T:

$$S_G=\frac{\text{s-value to reach NRI by }\mathrm{EI}^{0,1}}{\text{s-value to reach NRI by }\mathrm{EI}^{0,\lambda}}\quad,\quad S_T=\frac{S_G(\mathrm{NRI})}{\mathrm{RTF}}=S_G\times\frac{t_{WC}\text{ for the algorithm }\mathrm{EI}^{0,1}}{t_{WC}\text{ for the algorithm }\mathrm{EI}^{0,\lambda}}\ . \tag{3}$$

RTF is *a real time factor* which is the ratio of the corresponding wall clock times.

The performance of various algorithms can be seen in Figs. 1b-d. The results indicate that optimization makes slower progress w.r.t. the number of generations when the algorithms are asynchronous, when compared to its synchronous counterpart. Both synchronous and asynchronous algorithms know the true value of $p+s\times\lambda$ points at iteration s. However, at iteration s, the synchronous algorithm has obtained all the evaluations of the points it has created in the past, while the asynchronous algorithm has some of the points it generated in the past still being evaluated, with a risk of inaccuratly predicting their objective function. The synchronous algorithm always uses a complete information, i.e. both, the location and the objective function value, while the asynchronous method only avoids already searched regions, but it will often do it "blindly" without an available function value.

In order to have a more practical measure of the efficiency, one must incorporate the real time factors. A summary of the results generation and time-wise is presented in Table 2. In a characteristic manner, making $\mathrm{EI}^{0,1}$ asynchronous slows it down in terms of generations by a factor $1/S_G\approx 1/0.27\approx 3.7$ ("rank1approx9d"), but the real time speed up $S_T=2.9$ is notable. When $\lambda=4$, the slow-down w.r.t. the number of generations becomes less pronounced: considering the "rank1approx9d" problem, $1/S_G\approx 1/0.73\approx 1.4$, and the real time speed-up $S_T=5.8$ exceeds four. Therefore, asynchrony slows down the optimization progress w.r.t. the number of generations, but it reduces the t_{WC} values for an overall significant speed-up.

Including active points in the EI criterion ($\mu>0$) slightly improves the optimization speed-ups w.r.t. the number of generations. Note that it demands evaluating $\mu+\lambda$-dimensional integrals. The use of the Monte Carlo integration with larger numbers of samples increases the blocking time t_b substantially, which makes the generation of candidate solutions costly.

This work suggests as continuation the development of algorithms that will reduce the time needed to generate the λ candidate solutions, t_b. For example, one could parallelize the integration stage or, more generally, the generation of candidate solutions.

[1]The actual matrix is generated with the Scilab 5.3.3 "grand" function, the initial seed of the Mersenne Twister is 29.

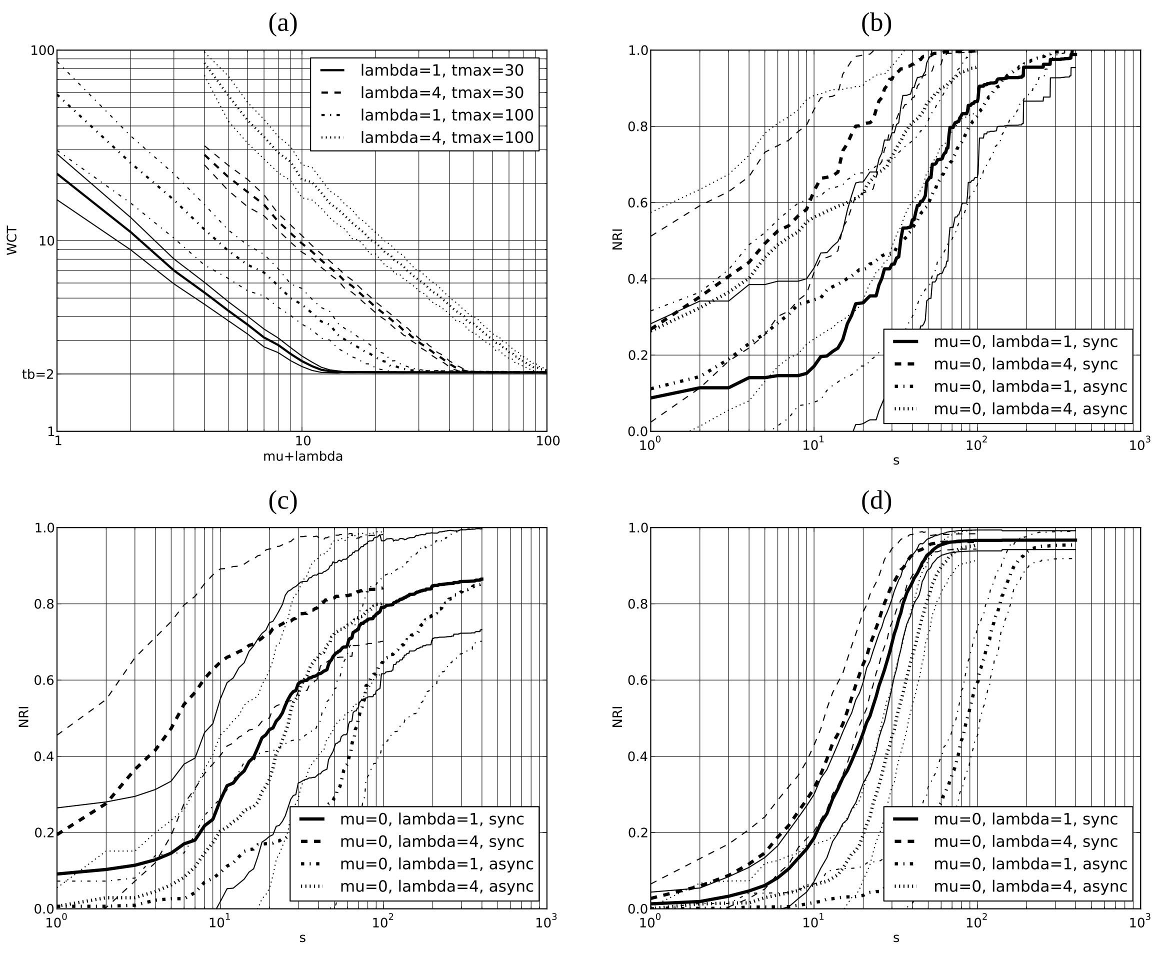

Figure 1: (a) Effect of the number of nodes, m, on wall clock times, $t_b = 2$, $t_{\min} = 10$ units. (b–d) Convergence of synchronous and asynchronous $\mathrm{EI}^{0,\lambda}$ optimizers in terms of generations, s : (b) "michalewicz2d", (c) "rosenbrock6d", (d) "rank1approx9d". In all the figures thicker lines correspond to the mean values, and the auxiliary thinner lines indicate the estimated deviations.

Table 2: Wall clock times, real time factors, and speed-ups of the EI algorithms. NRI $= 0.75$, $t_{\text{min}} = 10$, $t_{\text{max}} = 30$, $t_b = 2$. 100 repetitions, $m = 32$ nodes.

Acquisition	Node Access	t_{WC}	RTF	"michalewicz2d" S_G	"michalewicz2d" S_T	"rosenbrock6d" S_G	"rosenbrock6d" S_T	"rank1approx9d" S_G	"rank1approx9d" S_T
$\mathrm{EI}^{0,1}$	synchronous	22	1	1	1	1	1	1	1
$\mathrm{EI}^{0,4}$	synchronous	28	1.3	3.8	3.0	2.9	2.3	1.3	1.0
$\mathrm{EI}^{0,1}$	asynchronous	2.04	0.093	0.86	9.3	0.43	4.6	0.27	2.9
$\mathrm{EI}^{0,4}$	asynchronous	2.77	0.13	2.0	16	1.2	9.4	0.73	5.8
$\mathrm{EI}^{0,1}$	asynchronous	2.04	1	1	1	1	1	1	1
$\mathrm{EI}^{31,1}$	asynchronous	2.04	1	0.89	0.89	0.95	0.95	1.4	1.4
$\mathrm{EI}^{0,4}$	asynchronous	2.77	1.4	2.3	1.7	2.8	2.0	2.7	2.0
$\mathrm{EI}^{28,4}$	asynchronous	2.77	1.4	3.0	2.2	2.8	2.0	2.9	2.2

References

[1] J. Azimi, A. Jalali, and X. Z. Fern. Dynamic Batch Bayesian Optimization. NIPS, 2011.

[2] R. Girdziušas, R. Le Riche, F. Viale, and D. Ginsbourger. Parallel budgeted optimization applied to the design of an air duct. Technical report, École Nationale Supérieure des Mines de St-Étienne, 2010.

[3] ProActive PACA Grid. `http://proactive.inria.fr/`, 2012.

[4] D. Ginsbourger, R. Le Riche, and L. Carraro. Kriging is well-suited to parallelize optimization. In *chapter 6 of Computational Intelligence in Expensive Optimization Problems*, pages 131–162. Springer, April 2010.

[5] J. Azimi, A. Fern, and X. Z. Fern. Budgeted Optimization with Concurrent Stochastic-Duration Experiments. NIPS, 2011.

[6] J. Azimi, A. Jalali, and X. Z. Fern. Hybrid Batch Bayesian Optimization. ICML, 2012.

[7] T. Desautels, A. Krause, and J. Burdick. Parallelizing Exploration–Exploitation Tradeoffs with Gaussian Process Bandit Optimization. ICML, 2012.

[8] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. `arXiv:1206.2944v1`, 2012.

[9] M. Schonlau. *Computer Experiments and Global Optimization*. PhD thesis, Univ. of Waterloo, 1997.

[10] D. Ginsbourger, J. Janusevskis, and R. Le Riche. Dealing with asynchronicity in parallel Gaussian process-based global optimization. Technical report, École Nationale Supérieure des Mines de St-Étienne, 2010.

[11] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for Hyper-parameter Optimization. NIPS, 2011.

[12] Gene M. Amdhal. Validity of the single processor approach to achieving large-scale computing capabilities. In *AFIPS Spring Joint Computer Conference*, volume 30, pages 483–485, 1967.

[13] J. Janusevskis, R. Le Riche, D. Ginsbourger, and R. Girdziusas. Expected improvements for the asynchronous parallel global optimization of expensive functions: Potentials and challenges. In *LNCS 7219, selected articles from the LION 6 (Learning and Intelligent Optimization) Conf.* Springer, August 2012.

[14] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Comp.*, 11:1–18, 2003.