Analysis of a Batch Strategy for a Master-Worker Adaptive Selection Algorithm Framework

Christopher Jankee[1], Sébastien Verel[1], Bilel Derbel[2] and Cyril Fonlupt[1]
[1]*Université du Littoral Côte d'Opale, LISIC, France*
[2]*Université Lille 1, LIFL – CNRS – INRIA Lille, France*

Keywords: Master-Worker Architecture, Adaptive Selection Strategy.

Abstract: We look into the design of a parallel adaptive algorithm embedded in a master-slave scheme. The adaptive algorithm under study selects online and in parallel for each slave-node one algorithm from a portfolio. Indeed, many open questions still arise when designing an online distributed strategy that attributes optimally algorithms to distribute resources. We suggest to analyze the relevance of existing sequential adaptive strategies related to multi-armed bandits to the master-slave distributed framework. In particular, the comprehensive experimental study focuses on the gain of computing power, the adaptive ability of selection strategies, and the communication cost of the parallel system. In fact, we propose an adaptive batch mode in which a sequence of algorithms is submitted to each slave computing node to face a possibly high communication cost.

1 INTRODUCTION

In this work, we target black-box optimization problems for which no information can be known beforehand. There is no explicit hypothesis on these problems such as an explicit analytic expression of the fitness function, regularity, gradient properties, etc. This wide range of problems include optimization scenarios that require extensive numerical simulation. For example, some engineering design problems use a simulator of the physic to optimize criteria of interest for the designed system (Muniglia et al., 2016) (Armas et al., 2016). From a practical, as well as from a theoretical, point of view, in a black-box optimization scenario, the choice of the relevant algorithmic components according to the problem to solve is an open issue, and an active domain of research (Baudiš and Pošík, 2014). Moreover, when a large scale parallel compute environment is available, an optimization algorithm implies additional design challenges to make an effective cooperation between parallel resources. Thus, the paper investigates the online selection of algorithmic components in a master-worker framework in order to take full benefits from the compute power in a black-box optimization scenario.

From the early works of Grefenstette (Grefenstette, 1986) to recent works (Kotthoff, 0 30) and many other researchers (Kunanusont et al., 2017), the parameters setting or the algorithm selection, is a recurrent topic in evolutionary computation due to its crucial importance in practice. In this work, we are interested in adaptive algorithm selection; which consists in the online choice, among a number of alternatives stored beforehand in a portfolio, of an appropriate algorithm to execute next according to the current state of the search. Unlike the off-line tuning of parameters (Eiben et al., 2007) which selects an effective set of parameters (that is algorithmic components) before the execution of the optimization method, the online setting, also called *control* continuously selects an algorithm all along the optimization process using the feedback from the optimization algorithm being executed (Fialho et al., 2010; Baudiš and Pošík, 2014). Hence, online algorithm selection can be viewed as an adaptive optimization algorithm which follows the multi-armed bandit framework where the arms are the algorithms of the portfolio (DaCosta et al., 2008). The adaptive selection is then performed as follows. A reward is computed according to the performance observed when previously executing an algorithm.

As previously noticed, numerous black-box real world optimization problems, such as in engineering design, are computationally expensive, e.g., one fitness function evaluation can take several minutes (Muniglia et al., 2016). Thus, such problems can take benefit of the new compute facilities offered by large scale parallel platforms (clouds, pay-as-you-go, etc.). At the same time, it opens new research perspectives to develop original and more robust optimization methods. Several models of parallel evolutionary algorithms have been investigated for

Jankee C., Verel S., Derbel B. and Fonlupt C.
Analysis of a Batch Strategy for a Master-Worker Adaptive Selection Algorithm Framework.
DOI: 10.5220/0006504203130320
In *Proceedings of the 9th International Joint Conference on Computational Intelligence (IJCCI 2017)*, pages 313-320
ISBN: 978-989-758-274-5

parallel compute platform: from fine-grained (cellular model) to coarse grain (island) model (Tomassini, 2005). The centralized Master-Worker (M/W) architecture is the general research context of this work. Each worker computes a batch of actions scheduled by the master (i.e. mainly the evaluation of candidates solutions), and the master collects the local results from the workers, again mainly the fitness value of the best local solution, and coordinates the next actions to send to each worker. It is worth-noticing that this framework is often adopted in practice, not only due to the simplicity of deploying it over a real test bed, but also due to its high accuracy when dealing with computationally expensive optimization problems (Dasgupta and Michalewicz, 2013; Harada and Takadama, 2017).

In this context, we argue that an adaptive selection method of algorithms in a master-worker approach requires specific mechanisms in order to achieve optimal performances. In a sequential approach, the reward of each possible algorithm in the portfolio is updated according to the performance of the algorithm executed previously in the last iteration by one single process. In an M/W approach, the reward can be updated using the set of performances observed by the set of distributed workers. Nevertheless, two careful design components of the reward method has to be taken into account. First, if a batch of algorithms is executed by each worker node at each round, an accurate function to compute the local reward has to be defined. In addition, a global reward function is also required to aggregate the set of local rewards and to assign a reward of each algorithm from the portfolio. Besides, we can differentiate two types of parallel adaptive strategy: when adopting a homogenous strategy, all workers will execute the same algorithm at each round, or when adopting a heterogeneous strategy, the workers can execute different algorithms even in the case where a batch of algorithms is executed at each round by each worker. Several existing machine learning schemes have been used and studied previously in the sequential setting (Fialho et al., 2010), as well as in the decentralized island model (Derbel and Verel, 2011; Jankee et al., 2015). However, to our best knowledge, the analysis of online selection strategies have not been investigated within an M/W framework. Indeed, the M/W framework is more suitable to understand the tradeoff between an optimal global selection at the parallel system level and an optimal local selection of the most accurate algorithms at the worker level. In an M/W framework, the master node has a global view of the system which enables to define a global reward function and a global selection approach. This allows us to focus on the important selection strategy at the master level; and to propose new dedicated selection mechanisms. For instance, we deeply analyze a batch strategy which submits a sequence of actions to each worker (e.g. several fitness evaluation obtained by a set of algorithms). Such framework provides a way to deal with the tradeoff between communication and computation costs (fitness evaluation) as well as the adaptive efficiency of such strategy.

To summarize, we introduce a M/W algorithm selection framework contributing to the solving of the following questions:

(i) How to extend a selection strategy in a batch-oriented distributed framework ?

(ii) What is the impact of a batch framework on the performance of selection strategies ?

(iii) How does the batch framework affects the cost of communication?

Our M/W framework is evaluated using a tunable benchmark family and a simulation-based experimental procedure in order to abstract away the technical implementation issues, and instead provide a fundamental and comprehensive analysis of the expected empirical parallel performance of the underlying adaptive algorithm selection.

The rest of the paper is organized as follows. In Section 2, we review some related works.

In Section 3, the design components of our M/W adaptive framework is described in detail. In Section 4, we report our main experimental findings. In Section 5, we conclude the paper and discuss future research directions.

2 RELATED WORKS

In the following, we provide an overview of related studies on the algorithm selection problem in the sequential and distributed setting, as well as a brief summary of exiting optimization benchmark problems designed at the aim of evaluating their dynamics and behavior.

2.1 Sequential Adaptive Algorithm Selection

In the sequential setting, a number of reinforcement machine learning schemes have been proposed for the online and adaptive selection of algorithms from a given portfolio. Back to the early works of Grefenstette (Grefenstette, 1986), one standard technique consists in predicting the performance of a set of operators using a simple linear regression and the current

average fitness of the population, which then allows to select the best operator to be chosen according to the prediction given by the regression. However, recent works embed this selection problem into a multi-armed bandit framework dealing more explicitly with the tradeoff between the exploitation of the best so far identified algorithm, and the exploration of the remaining potentially underestimated algorithms.

A simple strategy is the so-called ε-greedy strategy which consists in selecting the algorithm with the best estimated performance at the rate $(1-\varepsilon)$, and a random one at rate ε.

The Upper Confidence Bound (UCB) strategy (Auer et al., 2002) is a state-of-the-art framework in machine-learning which consists in estimating the upper confidence bound of the expected reward of each arm by $\hat{\mu}_i + C \cdot e_i$; where $\hat{\mu}_i$ is the estimated (empirical) mean reward, and e_i is the standard error of the prediction. It then selects the algorithms with the higher bound (for maximization problem). The parameter C allows to tune the exploitation/exploration trade-off. In the context of algorithm selection (Fialho et al., 2010) where the arms could be neither independent nor stationary, the estimation of the expected reward is refined using a sliding window where only the W previous performance observations are considered.

The Adaptive Pursuit (AP) strategy (Thierens, 2005) is another technique using an exponential recency weighted average to estimate the expected reward with a parameter α to tune the adaptation rate of the estimation. This is used to define the probability p_i of selecting every algorithm from the portfolio. At each iteration, these probability values are updated according to a learning rate β, which basically allows to increase the selection probability for the best algorithm, and to decrease it for the other ones.

One key aspect to design a successful adaptive selection strategy is the estimation of the quality of an algorithm based on the observed rewards. Some authors showed that the maximum reward over a sliding window improves the performance compared to the means on some combinatorial problems (Fialho et al., 2010; Candan et al., 2013); but no fundamental analysis of this result was given. In genetic algorithms, the reward can be computed not only based on the quality but also on the diversity of the population (Maturana et al., 2009). In the context of parallel adaptive algorithm selection, the estimation of quality of each available algorithm is also a difficult question since not only one but many algorithms instances could be executed in each iteration.

2.2 Parallel Adaptive Algorithm Selection

The Master-Worker (M/W) architecture has been extensively studied in evolutionary computation (e.g., see (Dubreuil et al., 2006)). It is in fact simple to implement, and does not require sophisticated parallel operations.

Two communication modes are usually considered. In the synchronous mode, the distributed entities operate in rounds, where in each round the master communicates actions to the workers and then waits until receiving a response from every worker before starting a new round, and so on. In the asynchronous mode, the master does not need to wait for all workers; but instead can initiate a new communication with a worker, typically when that worker has terminated executing the previous action and is idle. When the evaluation time of the fitness function can vary substantially during the course of execution, the asynchronous mode is generally preferred (Yagoubi and Schoenauer, 2012) since it can substantially improve parallel efficiency. However, the synchronous mode can allow to have a more global view of the distributed system which can be crucially important to better coordinate the workers (Wessing et al., 2016).

Adaptive selection approaches designed to operate in a distributed setting are not new. The island model, which is considered as inherently distributed, has been investigated in the past. The first studies conducted in this context demonstrate that the dynamics of an optimal parallel selection method can be fundamentally different at the first sight from its sequential counterpart. For instance, because of the stochasticity of evolutionary operators, it has been noted that a set of heterogeneous nodes can outperform a set of nodes executing in parallel the same sequence of algorithms computed according to a sequential algorithm selection oracle (Derbel and Verel, 2011). To cite a few, in (Tanabe and Fukunaga, 2013; García-Valdez et al., 2014), it is also well known that a random setting the parameters at each iteration in a heterogeneous manner can outperform static homogeneous parameter settings. Nonetheless, embedding a reinforcement machine learning technique instead of random selection can improve the performance of the adaptive distributed system.

In (Derbel and Verel, 2011; Jankee et al., 2015), a distributed adaptive metaheuristic selection framework is proposed which can be viewed as a natural extension of the island model that was specifically designed to fit the distributed nature of the target compute platforms.

The adaptive selection is performed locally by

selecting the best rewarded metaheuristic from the neighboring nodes (islands) or a random one with small probability like in ε-greedy strategy. Notice, however, that we are not aware of any in-depth analysis addressing the design principles underlying an M/W adaptive algorithm selection approach. In this work, we propose and empirically analyze the behavior of such an approach in an attempt to fill the gap between the existing sequential algorithm selection methods and the possibility to deploy them in a parallel compute environment using a simple, yet effective, parallel scheme like the M/W one.

2.3 Benchmarks: The Fitness Cloud Model

The understanding of the dynamics of a selection strategy according to the problem at hand is a difficult issue. A number of artificial combinatorial problems have been designed and used in the literature. We can distinguish between two main benchmark classes. In the first one, a well-known combinatorial problem in evolutionary algorithm is used, such as oneMax or long-path problems, with basic operators, such as bit-flip, embedded in a $(1+\lambda)$-EA (DaCosta et al., 2008). This, however, can only highlight the search behavior according to few and problem-specific properties. In the second class of benchmarks, the problem and the stochastic operators are abstracted. The performance of each available operator is then defined according to the state of the search (Thierens, 2005; Fialho et al., 2010; Goëffon et al., 2016; Jankee et al., 2016). This allows to study important black box (problem independent) features such as the number of operators, the frequency of change of the best operators, the quality difference between operators, etc.

In this work, we use a tunable benchmark, called the Fitness Cloud Model (FCM), introduced recently in (Jankee et al., 2016). The FCM is a benchmark from the second class where the state of the search is given by the fitness of the solution. The fitness of a solution after applying a search operator is modeled by a random variable for which the probability distribution depends on the fitness of the current solution. A normal distribution with tunable parameters is typically used. More specifically, given the fitness $z = f(x)$ of the current solution x, the probability distribution of the fitness $f(y)$ of one solution obtained by a specific operator is defined by: $\Pr(f(y) = z' \mid f(x) = z) \sim \mathcal{N}(\mu(z), \sigma^2(z))$ where $\mu(z)$ and $\sigma^2(z)$ are respectively the mean and the variance of the normal distribution. In (Jankee et al., 2016), a simple scenario with two operators is studied. The mean and variance of the conditional normal distribution are defined as follows: $\mu_i(z) = z + K_{\mu_i}$ and $\sigma_i^2(z) = K_{\sigma_i}$ for each operator $i \in \{1,2\}$. Parameters K_{μ_i} and K_{σ_i} are different constant numbers. An adaptive algorithm is assumed to start with a search state where the fitness value is 0, and stops when a fitness value of 1 is reached. The expected running time of $(1+1)$-EA is then proved to be the inverse of the best expected improvement of among the considered operators, which can be analytically computed assuming a normal distribution. Notice that in the FCM, one can control the average quality and the variance of each operator as well the relative difference between the considered operators.

3 DESIGN BATCH ADAPTIVE MASTER-WORKER ALGORITHM

The global architecture of the proposed adaptive M/W framework is summarized in Algorithm 1. 2 depicts the high level code executed by the master. Algorithm 3 presents the high level code executed in parallel by each worker. The overall algorithm operates in different parallel rounds. At each round, the master sends the best solution $x^\star$ and a batch of operator identifiers Θ to be executed by each worker node i. After getting the batch, each worker executes iteratively its assigned operators starting from $x^\star$ as an initial solution and computes a new local best solution. The master waits for all best local solutions to be computed in parallel by the workers, and updates the $x^\star$ to be considered in the next round. The workers also inform back the master of the performance observed when executing the batch of operators. Note that when the size of the batch of operators is one, each worker executes only one fitness evaluation; which is basically the same as a synchronous parallel $(1+\lambda)$-EA with λ equals to the number of workers n.

In our algorithm selection setting, a portfolio of k (local search) operators is assumed to be given, and no a priori knowledge is assumed on the behavior of the operators. The adaptive part of our framework is mainly handled at the master level. In fact, after collecting the rewards computed locally by each worker, the master executes the main function Selection_Strategy (line 12 of Algo. 1) which allows him to compute the new set of batches Θ_i to be sent again to each worker i. Before going into further details, it is important to emphasize that the batch Θ_i is simply an ordered list of operators to be executed consecutively in a row by the corresponding worker i following the general framework of a $(1+1)$-EA as depicted in Al-

Algorithm 1: Adaptive M/W algorithm for the master node.

1: $(\Theta^1, \Theta^2, ..., \Theta^n) \leftarrow$Selection_Strategy_Initialization()
2: $x^\star \leftarrow$ Solution_Initialization() ; $f^\star \leftarrow f(x^\star)$
3: **repeat**
4: **for** each worker i **do**
5: Send Msg($\Theta^i, x^\star, f^\star$) to worker i
6: **end for**
7: Wait until all messages are received from all workers
8: **for** each worker i **do**
9: $(r^i, x^i, f^i) \leftarrow$ Receive Msg() from worker i
10: **end for**
11: $x^\star \leftarrow x^i; f^\star \leftarrow f^i$ s.t. $f^i = \max\{f^\star, f^1, f^2, \ldots, f^n\}$
12: $(\Theta^1, \Theta^2, ..., \Theta^n) \leftarrow$ Selection_Strategy$(r^1, ..., r^n)$
13: **until** stopping criterion is true

Algorithm 2: Adaptive algorithm selection strategy.

1: **function** SELECTION_STRATEGY($r^1, ..., r^n$)
2: **for** each operator j **do**
3: $R_j \leftarrow$ Global_Reward_Aggregation$(r^1, ..., r^n)$
4: **end for**
5: $(\Theta^1, \Theta^2, ..., \Theta^n) \leftarrow$Decision_Strategy$(R_1, R_2, ..., R_k)$
6: **return** $(\Theta^1, \Theta^2, ..., \Theta^n)$
7: **end function**

Algorithm 3 : Adaptive M/W algorithm for each worker node.

1: $(\Theta, x^\star, f^\star) \leftarrow$ Receive Msg() from master
2: **for** each index b of operators batch Θ **do**
3: $x' \leftarrow$ Apply operator Θ_b on $x^\star$
4: $f' \leftarrow$ Evaluate fitness of x'
5: $\delta_b \leftarrow \max(0, f' - f^\star)$
6: **if** $f(x^\star) < f(x')$ **then**
7: $x^\star \leftarrow x'$; $f^\star \leftarrow f'$
8: **end if**
9: **end for**
10: **for** each operator j **do**
11: $r_j \leftarrow$ Local_Reward_Aggregation(δ)
12: **end for**
13: Send Msg$((r_1, r_2, \ldots, r_k), x^*, f^*)$ to master

gorithm 3.

More precisely, the number of search iterations executed by each worker is the size of the batch, and the search operator applied at each iteration correspond to the order given in the same batch. At each iteration, the fitness improvement δ_b of each operator is evaluated (the positive fitness difference between solutions before and after applying the operator). The fitness improvement (Fialho et al., 2009) will allow each worker to compute a local reward for every executed operator. The Local_Reward_Aggregation function (line 11 of Algo. 3) allows each worker to compute a local reward r_j for each operator (if included in the batch) according to the observed improvements δ.

The main function Selection_Strategy executed at the master level is responsible for : (i) aggregating the local rewards sent by the worker and (ii) accordingly select the new set of operators batches. This is described in next subsection.

Table 1: Parameter use for the selection strategies.

Selection strategies	Parameter
UCB	$c = 0.005$, $w = 700$
AP	$\alpha = 0.2$, $\beta = 0.2$
ε-Greedy	$\varepsilon = 0.05$, $w = 4500$
δ-Greedy	$Inc = 10$

Operators	μ ($\times 10^{-4}$)	σ ($\times 10^{-4}$)
op_1	-1	1
op_2	-10	5

We also designed a simple heterogeneous selection strategy, called Δ-greedy. In the very first round, the operators are randomly assigned to workers just like for any other strategies. In the subsequent rounds, the number of workers associated with the best operator is increased by Δ, where Δ is an integer parameter; and the number of workers associated with the worst operator is decreased by Δ. The best (resp. worst) operator is computed as the one which attains the highest (resp. lowest) local reward in the previous round. In order to avoid side effects, the number of workers associated with each operator is bounded by a minimal number n_{min} which is the second parameter of this strategy. For each worker, the batch of operators remains, however, homogeneous for this strategy, i.e., a worker executes the same operator which is possibly different from the one executed by a different worker.

4 EXPERIMENTAL ANALYSIS

In this section, we analysis by experiments on the fitness cloud benchmark which defines a relevant scenario for M/S framework using a batch of operators the adaptive performance of difference selection strategies, and the parallel efficiency of using a batch of algorithms to execute on each worker according to the communication cost relatively to the computation cost of the fitness function.

4.1 Experimental Setup

In order to examine different possible parallel architecture, we choose a simulation-based approach where we count the number of round and the amount of communication performed by the master until reaching the optimal fitness value. In that way, we are able to discuss on the communication cost independently to a specific architecture. Following the Fitness Cloud Model (Jankee et al., 2016), we consider

a portfolio with two operators, both follows a normal distribution of fitness value as discussed in the section 2.3. The mean value of the first operator op_1 is chosen to be higher than the second operator op_2, but the variance of op_1 is smaller than the second operator. This scenario is one of the most relevant for a parallel system. Indeed, when one solution is created by an operator, the expected improvement of the first operator op_1 is higher than the one of the operators op_2. In that way, the first operator would be preferred in a sequential algorithm producing iteratively new solutions. On the contrary, when a number of solutions are created at the same time by an operator, the gain of the second operator op_2 is larger than the first operator, *i.e.* the longest tail of the fitness distribution of the second operator op_2 is more likely to produce a better solution. This scenario allows us to accurately analyze the impact of the number of workers and the size of the batch of algorithms send to each worker. The Table 1 gives the values of the parameters of the normal distribution used in this work.

In this work, the sequential selection strategies, AP, UCB, and ε-Greedy, (see Sect. 2.1) are tailored to a M/W framework. There are two main differences with the original versions. First, instead of one reward value, a set of reward values are used to update the quality of each operator. To this goal, the local and the global aggregation functions (see Algo. 1 and 3) compute the maximum of fitness improvements given, respectively, by the batch of operators and by the set of workers. Second, each selection strategy selects a set of operators instead of a single operator at each round. For UCB strategy which selects the operator with the highest score, the set of selected operators is homogeneous: for any workers and any operators from the batch of operators, the same operator (with the highest score) is selected. The other selection strategies (AP, ε-greedy, and δ-Greedy) defines a probability of selection of each operator. Then, for any operator in the batch of any worker, the operator is selected according to these probabilities. Those set of operators is heterogenous at the workers level and at the batch level.

The parameter set of the different selection strategies is given in Table 1. We follow the robust parameters proposed in (Jankee et al., 2016; Jankee et al., 2015). For all following experiments, the algorithm is repeated independently 32 times, and the average value of performance (number of rounds, or computation effort) are computed.

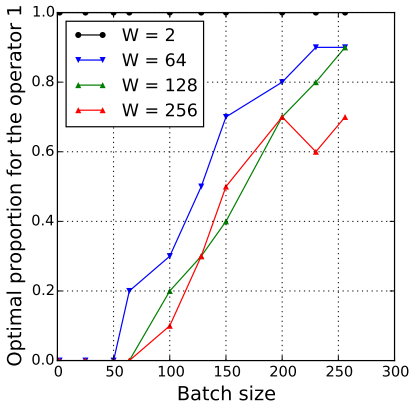

Figure 1: Random strategies in function the batch size and windows size, we observe the proportion to use the operator 2 that is to say the operator having long tail.

4.2 Baseline Selection Strategy

The operator op_1 and op_2 of the benchmark are static in the sense that the parameters of the fitness distribution does not change the optimization process according to the state of the search. In this case, a bias random selection of operators can be considered as a baseline strategy: with a rate p between 0 and 1, the operator op_1 is selected, and the with the rate $1-p$ the operator op_2 is selected. This baseline strategy helps to understand the tradeoff between a selection that promotes the worker level or that promotes the master level according to the operator batch size and the number of workers.

Figure 1 shows the value of p which obtains the best performance. This best value is computed off-line by experiments from the set $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$, and we denote by $p^\star$ this value. When the batch size is small, *i.e.* the number of operators in the batch send to each worker, except when the number of workers is very small (such as 2 workers), the optimal rate $p^\star$ is 0, and a bias random selection strategy always selects the second operator. In this case, the operator op_2 with the lowest average but longest tail is the most interesting one. On the contrary, the optimal rate $p^\star$ increases with the batch size. When the number of workers is equal to the size of batch, both equal to 128, the optimal bias random selection selects the operator 2 (with long tails) with rates 70% and therefore the operator 1 with rate 30% (with the highest average). The optimal $p^\star$ is larger than 80% when the batch size is large with respect to the number of workers (the batch size is the double of the number of workers). With lower amounts of information exchange between workers, an efficient search converges toward to sequential process where the first

operator with highest expected gain with one trial is preferred. In that case, a sequential setting in favor of the operator with a small improvement but with a high probability is promoted.

4.3 Adaptive Performance with a Batch Scheduling Technique

In Master/Worker framework, a batch scheduling technique can be used to reduce the communication to the computation cost. In this section, the goal is not to analyze the performance of such a technique from a purely parallel perspective, but instead, to highlight the accuracy of an adaptive strategy to select the relevant operators in the batch, which is a challenging issue *per se*. Instead of analyzing the number of rounds to optimum, we hence study the number of evaluations executed in parallel by all the workers. Since the range of the number of evaluations can be huge as a function of the number of workers and the batch size, we consider to normalize it by the average number of evaluations of the unbiased heterogeneous random strategy with $p = 0.5$ considered as a baseline. Due to lack of space, only a representative set of results is shown in Fig. 2, providing an overview of the relative behavior of the adaptive strategies with respect to the baseline one.

Notice that a strategy with a normalized value below 1 in Fig. 2 is better than the unbiased random one, and inversely. For completeness, we include the static strategies that would always select the same pre-fixed operator in all rounds.

For all adaptive strategies, the relative normalized performance decreases with the size of the batch scheduling. The adaptive strategies are in fact found to be relatively better than the random one for a small size batch, but, they become worse for a large size batch. On the contrary, the relative normalized performance of adaptive strategies increases with the number of workers. UCB and ε-greedy are not able to select the optimal operator in each round when the number of workers is lower than 50 (and batch size 128). In this case, the heterogeneous AP outperforms the other strategies, and can be better than an optimal homogeneous strategy. Actually, when the size of the batch is large with respect to the number of workers, the adaptive strategies fail to perform accurately. In this case, we hypothesize that alternative selection mechanism based on the local reward of each worker would be better than a selection based on the estimation of the global reward.

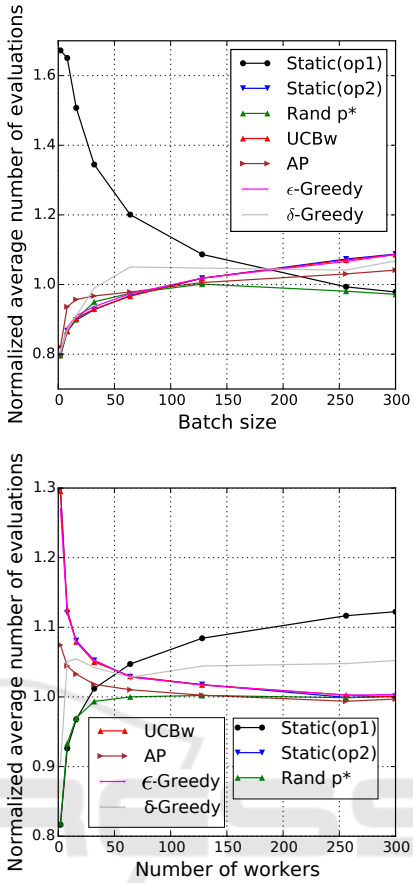

Figure 2: Normalized average number of evaluations as a function of batch size using 128 workers (top), and as a function of the number of workers using a batch size 128 (bottom). The values are normalized by the average number of evaluations of the heterogeneous random strategy with $p = 0.5$. The lower the better.

5 CONCLUSIONS

An efficient Master/Worker framework should deal with the tradeoff between the communication and the computation costs. Independently of the adaptive properties of the optimization method, a batch strategy can be used to reduce the communication cost but possibly decrease the performance of the optimization process. In this paper, we deeply analyze the adaptive method according to the number of workers as well of the size of the batch of operators sent to each worker. We show that another tradeoff is also required from an adaptive point of view, and the heterogeneous selection strategy outperforms homogeneous ones when the batch size is large with respect to the number of workers.

We proposed a naive batch strategy, the master de-

termines an operator list to be executed on the slave according to this knowledge. It would be interesting to have two selection strategies on the master, another on the worker. The master could send the parameters to the worker selection policy instead of pre-setting an operator list because we know that the operator to use is not necessarily the same according to the stage of the search state of a worker.

In another perspective, an asynchronous architecture would make it possible to improve the time of use of the slave processors, especially when the computation time of the fitness varies according to the state of the search. Moreover, the batch does not improve the synchronization between the different compute nodes. An asynchronous architecture makes it possible to better exploit the network resources.

REFERENCES

Armas, R., Aguirre, H., Zapotecas-Martínez, S., and Tanaka, K. (2016). Traffic signal optimization: Minimizing travel time and fuel consumption. In *EA 2015*, pages 29–43. Springer.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

Baudiš, P. and Pošík, P. (2014). Online black-box algorithm portfolios for continuous optimization. In *PPSN XIII*, pages 40–49. Springer.

Candan, C., Goëffon, A., Lardeux, F., and Saubion, F. (2013). Non stationary operator selection with island models. In *GECCO*, pages 1509–1516.

DaCosta, L., Fialho, A., Schoenauer, M., and Sebag, M. (2008). Adaptive operator selection with dynamic multi-armed bandits. In *GECCO*, page 913. ACM Press.

Dasgupta, D. and Michalewicz, Z. (2013). *Evolutionary algorithms in engineering applications*. Springer Science & Business Media.

Derbel, B. and Verel, S. (2011). DAMS: distributed adaptive metaheuristic selection. In *GECCO*, pages 1955–1962. ACM Press.

Dubreuil, M., Gagne, C., and Parizeau, M. (2006). Analysis of a master-slave architecture for distributed evolutionary computations. *IEEE T. on Systems, Man, and Cybernetics: Part B*, 36:229–235.

Eiben, A. E., Michalewicz, Z., Schoenauer, M., and Smith, J. E. (2007). Parameter control in evolutionary algorithms. In *Parameter Setting in Evolutionary Algorithms*, pages 19–46. Springer.

Fialho, A., Da Costa, L., Schoenauer, M., and Sebag, M. (2009). Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. In *LION'09*, volume 5851, pages 176–190. Springer.

Fialho, A., Da Costa, L., Schoenauer, M., and Sebag, M. (2010). Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60:25–64.

García-Valdez, M., Trujillo, L., Merelo-Guérvos, J. J., and Fernández-de Vega, F. (2014). Randomized parameter settings for heterogeneous workers in a pool-based evolutionary algorithm. In *PPSN XIII*, pages 702–710. Springer.

Goëffon, A., Lardeux, F., and Saubion, F. (2016). Simulating non-stationary operators in search algorithms. *Appl. Soft Comput.*, 38:257–268.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 16(1):122–128.

Harada, T. and Takadama, K. (2017). Performance comparison of parallel asynchronous multi-objective evolutionary algorithm with different asynchrony. In *CEC*.

Jankee, C., Verel, S., Derbel, B., and Fonlupt, C. (2015). Distributed Adaptive Metaheuristic Selection: Comparisons of Selection Strategies. In *EA 2015*, pages 83–96.

Jankee, C., Verel, S., Derbel, B., and Fonlupt, C. (2016). A fitness cloud model for adaptive metaheuristic selection methods. In *PPSN 2016*, pages 80–90. Springer.

Kotthoff, L. (2012-10-30). Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, pages 48–60.

Kunanusont, K., Gaina, R. D., Liu, J., Perez-Liebana, D., and Lucas, S. M. (2017). The n-tuple bandit evolutionary algorithm for automatic game improvement. In *CEC*.

Maturana, J., Fialho, Á., Saubion, F., Schoenauer, M., and Sebag, M. (2009). Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In *CEC*, pages 365–372. IEEE.

Muniglia, M., Do, J.-M., Jean-Charles, L. P., Grard, H., Verel, S., and David, S. (2016). A Multi-Physics PWR Model for the Load Following. In *ICAPP*.

Tanabe, R. and Fukunaga, A. (2013). Evaluation of a randomized parameter setting strategy for island-model evolutionary algorithms. In *CEC 2013*, pages 1263–1270.

Thierens, D. (2005). An adaptive pursuit strategy for allocating operator probabilities. In *GECCO'05*, pages 1539–1546.

Tomassini, M. (2005). *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Wessing, S., Rudolph, G., and Menges, D. A. (2016). Comparing asynchronous and synchronous parallelization of the sms-emoa. In *PPSN XIV*, pages 558–567, Cham. Springer.

Yagoubi, M. and Schoenauer, M. (2012). Asynchronous master/slave moeas and heterogeneous evaluation costs. In *GECCO*, pages 1007–1014.