Combining Local Surrogates and Adaptive Restarts for Global Optimization of Moderately Expensive Functions

Taimoor Akhtar[1, a)] and Christine A. Shoemaker[2, 3, b)]

[1]*NUS Environmental Research Institute (NERI), National University of Singapore.*
[2]*Department of Industrial Systems Engineering and Management, National University of Singapore.*
[3]*Department of Civil and Environmental Engineering, National University of Singapore.*

a) Corresponding author: erita@nus.edu.sg
b) shoemaker@nus.edu.sg

Abstract. This study combines the ideas of using local surrogates and a restart mechanism to improve the algorithm runtime and optimization efficiency of iterative surrogate global optimization frameworks. The proposed framework, LSOR (**L**ocal Response Surface based **S**urrogate **O**ptimization with Adaptive **R**estarts), is a modification of the DYCORS framework proposed by Regis and Shoemaker [1], and uses locally fitted RBF (Radial Basis Function) surrogates, candidate search and restarts during the optimization. The key purpose of LSOR is to enable many function evaluations without increasing algorithm run-time complexity. Hence, LSOR is suitable for parallel optimization settings and for moderately expensive problems. Numerical results on ten test problems with a budget of 5000 evaluations show that LSOR has comparable accuracy to DYCORS with a significantly lower algorithm runtime.

BACKGROUND AND MOTIVATION

Many real-world applications use continuous global optimization algorithms to solve computationally expensive multi-modal simulation-optimization problems [2]. Global methods for solving such problems include deterministic mathematical programming algorithms [2-5] and stochastic metaheuristics [6-9]. A recent study [10] shows that both deterministic methods and stochastic metaheuristics are effective and compare well on different evaluation budgets.

Iterative surrogate algorithms are stochastic metaheuristics that are effective in solving computationally expensive multi-modal problems. Past work on iterative surrogate optimization has mostly focused on designing algorithms for a limited evaluation budget (e.g., a few hundred function evaluations). Research in this area is dominated by algorithms that use Gaussian Processes [6,7] and Radial Basis Functions [1,8,9] as global surrogates (i.e. surrogates that cover the entire domain). Many simulation-optimization problems, however, may require more than a few hundred function evaluations (even with surrogates). Such problems include moderately expensive functions (e.g., evaluation time is seconds or less) or problems within parallel optimization frameworks [3] that require a large evaluation budget (e.g., a few thousand evaluations). Fitting a global surrogate (i.e., using all previously evaluated points) when evaluation budget is large may not be computationally feasible (since fitting the surrogate is of order $O(n^2)$ at least, for n evaluated points). This study explores the use of locally fitted surrogates and adaptive restarts, and presents a stochastic surrogate framework, LSOR, designed for global optimization with a relatively large evaluation budget (in thousands).

THE LSOR METHODOLOGY

Overview of the LSOR Framework

The LSOR framework is derived from the DYCORS algorithm, by Regis and Shoemaker [1]. The DYCORS algorithm is an iterative / sequential surrogate optimization method for finding the global optimum of a multimodal function. At each iteration of the algorithm, DYCORS employs an RBF surrogate to propose and evaluate new expensive points, and the RBF surrogate is updated subsequently (using all points evaluated so far). Many candidate points are generated at each algorithm iteration via random perturbation of a subset of decision variables around the best point evaluated so far. Some of these candidates are selected for expensive evaluations via use of a weighted score / acquisition function that balances exploration and exploitation. This is effective for high-dimensional problems. DYCORS also has a restart mechanism that triggers if search fails to improve too many times.

A potential drawback of the DYCORS framework, in terms of algorithm runtime, is that fitting of the surrogate model may become computationally expensive (e.g. at least $O(n^2)$ when the number of previously evaluated points n is large. This significantly affects the algorithm runtime. The runtime efficiency improves via triggered restarts. However, in this study, we modify the framework further and propose a mechanism for fitting local surrogates around the best point, at each algorithm iteration (using a subset of evaluated points). The resulting framework, i.e., LSOR (see Figure 1 for overview) is more efficient in terms of runtime.

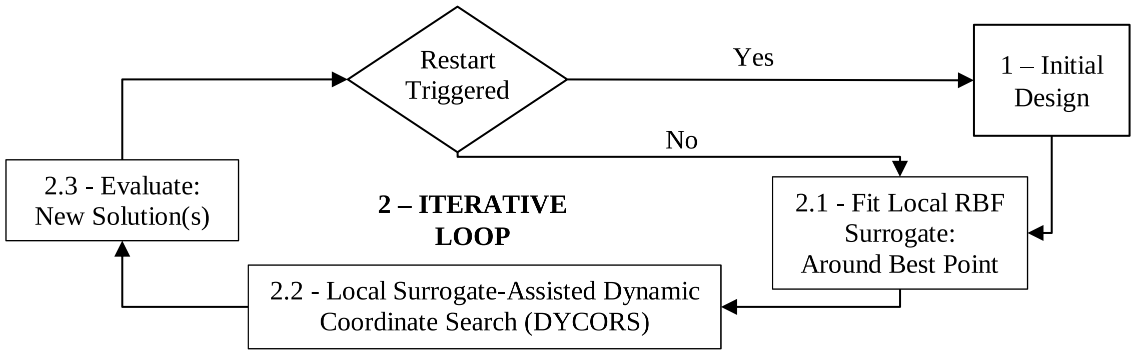
Restart Triggered
Yes
1 – Initial Design
No
2.3 - Evaluate: New Solution(s)
2 – ITERATIVE LOOP
2.1 - Fit Local RBF Surrogate: Around Best Point
2.2 - Local Surrogate-Assisted Dynamic Coordinate Search (DYCORS)

FIGURE 1. The LSOR Optimization framework.

Local Surrogate Fitting and LSOR Variants

We fit a Local RBF surrogate around the best point found so far (since candidates are also generated by perturbing the best point) at each iteration of LSOR (see Step 2.1 in Figure 1). The local RBF surrogate is fitted by using $\mathbf{N_{rbf}}$ points that are closest to (and including) the current best point, $\mathbf{X_{best}}$, in terms of Euclidean distance. Since we use an RBF surrogate with a polynomial tail, a minimum of **2d+2** (where **d** is the number of decision variables) points are required for surrogate fitting [8] and we propose to use only **2d+2** points for local surrogate fitting in LSOR.

Another modification from DYCORS that we propose in LSOR is the reduction in the number of candidate (trial) points that are generated at each algorithm iteration (Step 2.2 in Figure 1). DYCORS generates 100*d candidate points at each algorithm iteration. However, surrogate evaluation and determination of weighted score / acquisition for a large number of candidates may also require significant computation time. Hence, in LSOR, we also propose a reduction of number of candidate points generated to 10*d. In order to analyze the individual impacts of fitting local surrogates and reducing the number of candidates generated, we propose two alternative variants of LSOR and DYCORS, and analyze them in our computational experiments. We call these variants LSOR-L and DYCORS-C, respectively. Parameter settings for DYCORS, DYCORS-C, LSOR-L and LSOR are given in Table 1. We can see from Table 1 that DYCORS-C is like DYCORS (i.e., a global surrogate is used) except the number of candidate points is reduced to 10*d. LSOR-L is like DYCORS except that a local surrogate is used, and LSOR is like LSOR-L except the number of candidates is reduced to 10*d in LSOR.

TABLE 1. Algorithm parameter settings for DYCORS and the three variants of LSOR. N_{cand} is the number of candidates generated at each algorithm iteration and N_{budget} is the total budget of function evaluations.

Algorithm Name	Number of Points in Surrogate	Number of Candidates in Search
DYCORS	Global: $N_{rbf} = N_{budget}$	$N_{cand} = 100*d$
DYCORS-C	Global: $N_{rbf} = N_{budget}$	$N_{cand} = 10*d$
LSOR-L	$N_{rbf} = 2*d+2$	$N_{cand} = 100*d$
LSOR	$N_{rbf} = 2*d+2$	$N_{cand} = 10*d$

RESULTS AND DISCUSSION

Experimental Setup

Test Problems and General Experimental Setup: We test LSOR and DYCORS on ten noiseless BBOB benchmark [11] problems (F_{15}-F_{24}). We chose F_{15}-F_{24} since they are multi-modal. Number of decision variables (can vary) are set to ten for all test problems. Evaluation budget for all test problems is limited to 5000 and we run ten trials for each experiment. Both DYCORS and LSOR propose four function evaluations at each algorithm iteration.

Algorithm Runtime Analysis: A primary purpose of the LSOR framework is to reduce the runtime of surrogate optimization, and to ensure that running time of each algorithm iteration does not increase exponentially with number of function evaluations. Hence, we analyze the *total runtime* and the *runtime per iteration* of all algorithms. Both algorithms are thus implemented in the same Python optimization toolbox, called PySOT [12]. All experiments (recording runtime statistics) are performed on the same quadcore desktop machine.

Algorithm Performance Analysis: Performance of algorithms is compared in terms of progress plots, i.e., plot of best objective value against number of function evaluations. In this paper, we show progress plots of Problems F_{15} and F_{16} only. However, results of other test problems are similar. We do not compare against other algorithms in this analysis, since previous papers have shown that DYCORS is more efficient than numerous algorithms, including Genetic Algorithms and Bayesian Optimization methods, on test problems [1] and real world problems [13].

Results: Progress Plots

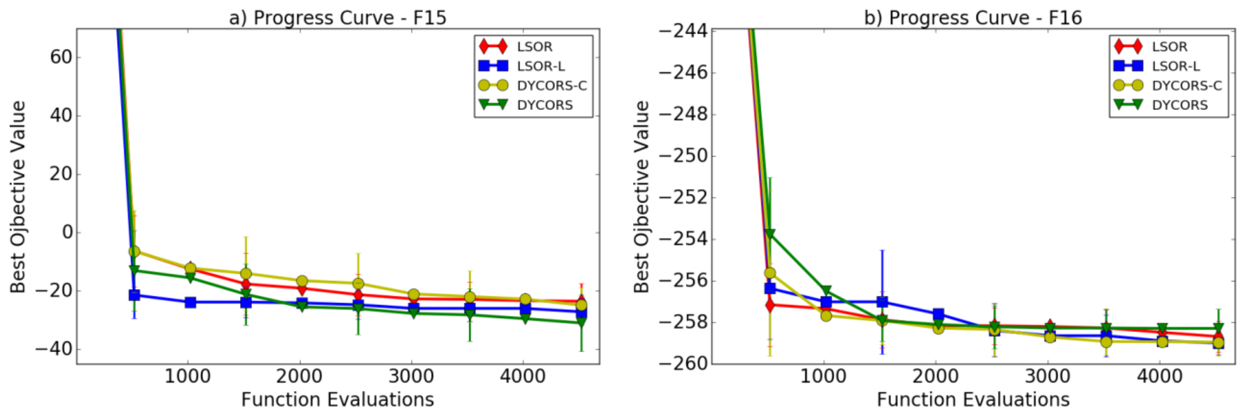

FIGURE 2. Progress plot comparisons, i.e. plots of best objective value vs functions evaluations (low curves are better) for: a) Problem F15 and b) Problem F16.

The progress curves of Fig. 2 show that all algorithms have a very similar rate of convergence for both problems F15 and F16. In case of Problem F15, DYCORS is marginally better than DYCORS-C and the LSOR variants after 5000 function evaluations. However, for Problem F16 (Fig. 3b), DYCORS-C and the LSOR variants perform slightly better. The difference in performance is not significant though. Results for the other 8 test problems are similar. Hence, it can be concluded that LSOR's performance in terms of function evaluations is comparable to DYCORS.

Results: Algorithm Runtime Analysis

While Figure 2 indicates that algorithmic performance (in terms of convergence per function evaluations) of LSOR is comparable to DYCORS, it is important to analyze whether LSOR has a better runtime in comparison to DYCORS. Figure 3 provides a summary of this analysis in terms of *runtime per algorithm iteration* (Fig. 3a) and *total algorithm runtime per experiment* (Fig. 3b), for Problem F15.

It is evident from Fig.3 that runtime per iteration (Fig. 3a) of LSOR is considerably lower than that of DYCORS, indicating that algorithm runtime efficiency is significantly improved by fitting local surrogates. Furthermore, the runtime per iteration of LSOR and LSOR-L remain constant as number of function evaluations and iterations increase. This indicates that algorithm runtime for LSOR and LSOR-L increases linearly with the number of evaluations. This is not the case for DYCORS and DYCORS-C. However, even for DYCORS, due to the restart mechanism, algorithm runtime per iteration does not increase exponentially. Overall (see Fig. 3b), LSOR's total runtime (average) is approximately five times lower than DYCORS for the F15 problem.

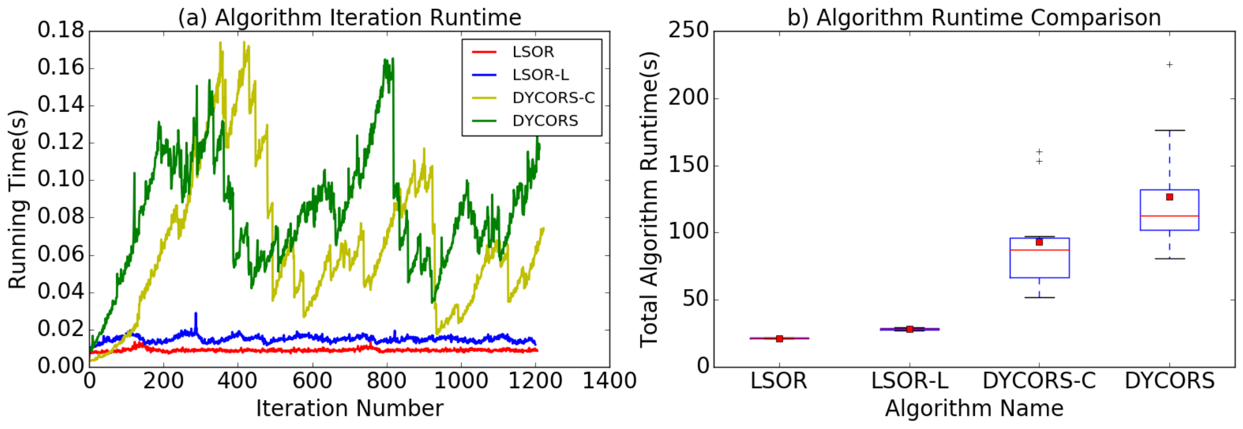

FIGURE 3. Comparison of runtime of each algorithm variant from Table 1 for problem F_{15}: a) Visualisation of average runtime (over multiple trials) for each iteration of all algorithms vs the iteration number, b) Boxplots of total runtime (multiple trials) of each algorithm for a budget of 5000 evaluations (LSOR has lowest runtime). Low runtimes are best.

CONCLUSION

LSOR is a local surrogate optimization framework with restarts, designed to efficiently run with a large evaluation budget. We compared algorithmic and runtime performance of LSOR with DYCORS. Results on test functions indicate that LSOR's performance (in terms of speed of convergence) is comparable to DYCORS and the runtime is significantly better. In future, we plan to extend LSOR to run in a large-scale parallel framework.

ACKNOWLEDGMENTS

This work is funded by Phase 2 of Energy and Environmental Sustainability for Megacities (E2S2) CREATE research program and Prof. Shoemaker's startup grant from National University of Singapore (NUS). LSOR and DYCORS are implemented in the PySOT [12] toolbox.

REFERENCES

1. R. G. Regis and C. A. Shoemaker, Engineering Optimization **45**, 529–555 (2013).
2. J. D. Pintér, *Global Optimization in Action* (Kluwer Academic Publishers, Dordrecht, Netherlands 1996).
3. R. G. Strongin and Y. D. Sergeyev, *Global Optimization with Non-Convex Constraints - Sequential and Parallel Algorithms* (Springer-Verlag, Berlin, Heidelberg, 2000).
4. Y. D. Sergeyev and D. E. Kvasov, *Deterministic Global Optimization* (Springer-Verlag, New York, 2017).
5. R. Paulavičius, Y. D. Sergeyev, D. E. Kvasov and J. Žilinskas, J. Glob. Optim. **59**, 545–567 (2014).
6. J. Snoek, H. Larochlle and R. P Adams, Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS), Nevada 2012, Curran Associates Inc., Vol. 2, pp. 2951-2959.
7. D. R. Jones, M. Schonlau and W. J. Welch, J. Glob. Optim. **13**, 455–492 (1998).
8. R. G. Regis and C. A. Shoemaker, INFORMS J on Comp. **19**, 497–509 (2007).
9. T. Krityakierne, T. Akhtar and C. A. Shoemaker, J. Glob. Optim. **66**, 417–437 (2016).
10. Y. D. Sergeyev, D. E. Kvasov and M. S. Mukhametzhanov, Scientific Reports. **8**, 453 (2018).
11. N. Hansen, S. Finck, R. Ros, and A. Auger, Technical Report RR-6829, INRIA (2009).
12. D. Eriksson, D. Bindel and C. A. Shoemaker, Surrogate optimization toolbox, 2015, see https://github.com/dme65/pySOT.
13. I. Ilievski, T. Akhtar, J. Feng, C. A. Shoemaker, Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17), California 2017, AAAI Publication, pp. 822-829.