EMPIRICALLY COMPARING THE FINITE-TIME PERFORMANCE OF SIMULATION-OPTIMIZATION ALGORITHMS

Naijia (Anna) Dong
David J. Eckman
Xueqi Zhao
Shane G. Henderson

Matthias Poloczek

School of Operations Research and Info. Engrg
Cornell University
206 Rhodes Hall
Ithaca, NY 14853, USA

Department of Systems and Indust. Engrg
University of Arizona
1127 East James E. Rogers Way
Tucson, AZ 85721, USA

ABSTRACT

We empirically evaluate the finite-time performance of several simulation-optimization algorithms on a testbed of problems with the goal of motivating further development of algorithms with strong finite-time performance. We investigate if the observed performance of the algorithms can be explained by properties of the problems, e.g., the number of decision variables, the topology of the objective function, or the magnitude of the simulation error.

1 INTRODUCTION

The practice of simulation optimization (SO) deals with optimizing a real-valued objective function that cannot be evaluated exactly, but instead must be estimated via simulation. In addition to the challenge of estimating the objective function, its structural properties, e.g., continuity, differentiability, or convexity, may be unknown. On account of this, SO algorithms are often designed to solve a broad class of problems without exploiting any structure of the objective function.

In the SO literature, one commonly sees theoretical results on the asymptotic performance of an algorithm. It is often shown that an algorithm will converge to a local or global optimizer as the simulation effort approaches infinity. Some results further specify a rate at which an algorithm converges once within a neighborhood of an optimizer. Unfortunately, the asymptotic regimes in which these results hold likely require an amount of computational effort that exceeds practical budgets, making the asymptotic results less useful to practitioners. For a practitioner deciding which algorithm to use for a particular problem, understanding how an algorithm can be expected to perform for a finite simulation budget is more meaningful.

The SO community lags behind other research communities when it comes to having an established testbed of problems and developing metrics for comparing the empirical finite-time performance of algorithms (Pasupathy and Henderson 2006). Moreover, a comprehensive comparison of SO algorithms has not been done on a large testbed (Amaran et al. 2016). As a first step toward such a comparison, we implement several popular SO algorithms and test them on a subset of problems from `SimOpt` (SimOpt Library 2011), a growing library of SO problems developed by Pasupathy and Henderson (2011). We evaluate the finite-time performance of the algorithms and discuss insights into the types of problems on which the algorithms might be expected to perform well. An objective of our study is to spur the development of SO algorithms that have strong finite-time performance for various classes of problems. We also hope to encourage further contributions to the `SimOpt` library so that a more comprehensive testbed of SO problems can become available to researchers for evaluating new algorithms.

2 EVALUATING FINITE-TIME PERFORMANCE

In deterministic optimization, an algorithm's performance on a given problem is usually measured by either the number of function evaluations or the wall-clock time needed to find the optimal solution or to get within a specified tolerance. In this way, one can easily compare algorithms. Applying this approach to SO algorithms runs into several challenges. Firstly, the optimal solutions to SO problems are often unknown or have no certificate of optimality. Secondly, sampling error makes it harder to determine whether the objective function value of a solution is within a given tolerance of the optimal objective function value. Instead, SO algorithms can be more fairly compared by fixing a given simulation budget—either the wall-clock time or the number of objective function evaluations—and evaluating the objective function at the estimated best solution visited within the budget.

Although measuring a simulation budget in terms of wall-clock time may make sense from a practical standpoint, the resulting performances are platform dependent. On the other hand, using the number of objective function evaluations as a measure of time has its own potential issues; see also Pasupathy and Henderson (2006). Firstly, a potentially significant portion of computational effort may go unmeasured if, for example, the constraints are stochastic or gradient information is calculated without taking additional objective function valuations. Secondly, for steady-state simulations that involve simulating a single, long sample path, counting one replication as one objective function evaluation may be misleading.

In our experiments, we specify the simulation budget in terms of the number of objective function evaluations, i.e., replications. Our justification for this choice is twofold: all of the problems we study have deterministic constraints and the simulations all have finite horizons. Moreover, we count all objective function evaluations towards the budget, including function evaluations used to obtain gradient estimates, e.g., via finite differences; see also Sect. 3.1.

For evaluating finite-time performance, we fix a simulation budget and consider the estimated best system visited before the budget is exhausted. On a given macroreplication, i.e., a single execution of an algorithm on a given problem instance that exhausts the budget, let $Z(n)$ denote the *true* objective function value of the *estimated* best solution visited in the first n objective function evaluations. Because the estimated best solution $X(n)$ is random, $Z(n)$ is a random variable. Conditional on the solution $X(n)$, the objective value $Z(n)$ is not random, but we probably cannot compute it exactly because we use simulation to evaluate the objective function. In our experiments, we get fairly precise estimates of $Z(n)$—conditional on $X(n)$—by running additional replications in a post-processing step. These replications are not counted towards the algorithm's budget.

Plotting $Z(n)$ as a function of n shows how the objective value of the estimated best solution changes as the algorithm progresses (Pasupathy and Henderson 2006). For most of the algorithms we test, they use the budget only to determine when to stop, and not in setting other algorithm parameters. For such algorithms, the random variable $Z(n)$ for fixed n is equivalent in distribution to the objective function value of the best system visited when running the algorithm with a simulation budget of n function evaluations. This implies that by fixing a large simulation budget, a single macroreplication can be used to evaluate an algorithm's finite-time performance for a range of budget values. The lone exception in our study is SPSA which takes the simulation budget as an input for setting its parameters. For SPSA, it is necessary to rerun the algorithm with different budgets in order to evaluate its performance.

Plotting the $Z(n)$ curve for one macroreplication has limited value since the location of the curve is itself random. Instead, it is more informative to run multiple macroreplications and average to obtain a mean performance curve $\bar{Z}(n)$. The *median* performance of an algorithm is also worthy of consideration as it is less sensitive to outliers in performance. The empirical cumulative distribution function (cdf) of $Z(n)$ also contains a great deal of information about how an algorithm performs, including its variability. Although plotting the empirical cdf of $Z(n)$ for a single algorithm on a single problem is straightforward, comparing empirical cdfs for multiple algorithms is challenging.

By testing algorithms on only a modest number of SO problems, we were able to present in this paper many of the plots of $\bar{Z}(n)$. Ideally, we would like to graphically compare the performances of

algorithms *across* a large number of problems as is done in the deterministic optimization community using performance profiles (Dolan and Moré 2002). Adapting performance profiles to simulation optimization remains an active area of research with a notable challenge being how exactly to define the performance ratio. See Ali, Khompatraporn, and Zabinsky (2005) and Rios and Sahinidis (2013) for examples of how performance profiles have been adapted to different classes of optimization problems. We believe that performance profiles have great potential for future comparisons of SO algorithms, especially when a large testbed of problems is available.

3 ALGORITHMS AND PROBLEMS

3.1 Algorithms

We suppose that when evaluating a solution x, an algorithm observes $y(x) = f(x) + \varepsilon(x)$ where $f(x)$ is the value of the objective function f at x and $\varepsilon(x)$ is the observational noise associated with simulating x. We further assume that $\varepsilon(x)$ has a mean of zero and finite variance. In our implementations, each algorithm estimates the objective value at a given solution by taking 30 replications and using the sample mean $\bar{y}(x)$.

All problems specify a deterministic domain $\mathscr{D} \subseteq \mathbb{R}^d$ where d is the number of decision variables. Unless otherwise stated, the initial solution x_1 is drawn from within $\mathscr{D}$ according to some probability distribution. For problems with bounded domains, we used a uniform distribution and for unbounded domains, we used either independent exponential or Laplace distributions for generating the components of x_1. The exact parameters of these distributions are given in the source code posted in our public repository (Bitbucket 2017). If an algorithm attempts to visit a solution $x_{k+1} \notin \mathscr{D}$, it instead samples the solution located where the the line connecting x_{k+1} and the previous solution x_k intersects the boundary of $\mathscr{D}$.

For our experiments, we selected a variety of well-known SO algorithms and compared them with two baseline methods: random search and gradient-based search. We provide a high-level description of the algorithms here and refer the interested reader to our repository for the codes.

Random Search

Random search (`RandomSearch`) iteratively evaluates solutions until the simulation budget is exhausted. The solutions are drawn from $\mathscr{D}$ according to the same probability distribution used to draw x_1 and the solutions are independent and identically distributed.

Gradient Search with Random Restarts

The gradient-based search algorithm (`GradSearch`) approximates the gradient of the objective function at a given solution x_k via d finite differences. That is, the jth component of the approximation $\hat{g}(x_k)$ is

$$\hat{g}_j(x_k) = \frac{\bar{y}(x_k + h_j e_j) - \bar{y}(x_k - h_j e_j)}{2h_j},$$

where for iterations $k \geq 2$, $h_j := \sqrt{\text{Var}(y(x_1))}/(\sqrt{2(30)} \cdot \hat{g}_j(x_1))$, $\text{Var}(y(x_1))$ is the *estimated* variance of the noise $\varepsilon(x_1)$, we perform 30 replications, and e_j is the d-dimensional unit vector whose jth entry is one. In the first iteration, we randomly draw two solutions x_1 and x' from $\mathscr{D}$ and set $h_j = \min_i \|x_1[i] - x'[i]\|/3$ for all $j \in \{1, \dots, d\}$, in order to compute $\hat{g}_j(x_1)$. In a given iteration k, we first test a step size of $c_k = 2$ by evaluating $\bar{y}(x_k + \hat{g}_k \cdot c_k)$ (for maximization problems). If this yields a better solution than x_k, we set $x_{k+1} = x_k + \hat{g}_k \cdot c_k$. Otherwise we iteratively divide c_k by 2 and test again, until either a better solution is found or c_k is too small, in which case we choose x_{k+1} to be a random point.

In order to prevent `GradSearch` from becoming trapped at a local optimum, the algorithm restarts from a randomly chosen solution if *all* of the following conditions are met:

  i) $\bar{y}(x_k) - \bar{y}(x_{k+1}) < \tau \cdot (1 + \|\bar{y}(x_{k+1})\|)$,   ii) $\|x_k - x_{k+1}\| < \sqrt{\tau} \cdot (1 + \|x_{k+1}\|)$,

iii) $\|\hat{g}_k\| < \sqrt[3]{\tau} \cdot (1 + \|\bar{y}(x_{k+1})\|)$, and   iv) $\|\hat{g}_k\| < \text{Var}(y(x_k))$,

where τ is a constant chosen by the user. In our experiments, we set $\tau = 10^{-4}$.

The Simultaneous Perturbation Method

The Simultaneous Perturbation Method (SPSA) (Spall 1992, Spall 1998) also performs an iterative search that starts from a randomly selected solution x_1 and approximates the gradient in each step. However, instead of using the method of finite differences, SPSA relies on a simultaneous perturbation approximation of the gradient: the jth component of $g(x_k)$ is approximated as

$$\hat{g}_j(x_k) = \frac{\bar{y}(x_k + c_k\Delta_k) - \bar{y}(x_k - c_k\Delta_k)}{2c_k\Delta_k(j)},$$

where Δ_k is a carefully chosen d-dimensional random vector; see Spall (1998) for details. SPSA requires only $2 \cdot 30 = 60$ function evaluations per iteration to approximate the gradient, whereas GradSearch requires $60d$. For the step-length sequence, we used the "automatic gain selection" implementation of Spall (2001).

The Stochastic Trust-Region Response-Surface Method

The Stochastic Trust-Region Response-Surface Method (STRONG) of Chang, Hong, and Wan (2013) approximates the unknown objective function by a series of local models, where each model is sufficiently accurate within its corresponding *trust region*. In iteration k, STRONG constructs a local model r_k that is believed to resemble the objective function within a trust region $\mathscr{B}(x_k, \Delta_k)$ of radius Δ_k centered at x_k. It then computes a point $x_k^* \in \mathscr{B}(x_k, \Delta_k)$ that is "close" to the optimum of r_k over $\mathscr{B}(x_k, \Delta_k)$. The point x_k^* is accepted and becomes the center point x_{k+1} for the next iteration if the following two tests are both passed:

1. The estimated improvement in objective value $\bar{y}(x_k) - \bar{y}(x_k^*)$ (for minimization problems) is sufficiently large compared to the predicted improvement $r_k(x_k) - r_k(x_k^*)$.
2. The improvement in objective value is statistically significant, accounting for observational noise.

If the local model fits the observed data at x_k and x_k^* well—indicated by satisfactory results to the above tests—then the radius of the trust region for the next iteration is increased. Otherwise it stays the same or, in the case of a poor prediction, is decreased.

Our implementation of STRONG follows Chang, Hong, and Wan (2013), with the exception that we do not apply design of experiments to select the evaluation points for fitting. Instead, we use central finite differences to estimate the gradient and a BFGS update to estimate the Hessian. We also tested a version (STRONGstg1) that does not fit second-order models, i.e., it only applies Stage I of STRONG.

The Nelder-Mead Algorithm

The algorithm of Nelder and Mead (1965) (Nelder-Mead) iteratively maintains a simplex of $d+1$ vertices whose centroid is denoted by x_k. In iteration k, the vertex with the worst observed objective value, say z_k, is reflected through the centroid of the remaining d vertices to obtain a new point z_k' that is then sampled. If the observed value $\bar{y}(z_k')$ is worse than the values previously observed at the remaining d vertices of the simplex, the simplex is contracted and z_k' is chosen closer to the centroid. Otherwise z_k is removed from the simplex and z_k' is added as $(d+1)$st vertex. We implemented the improvements suggested by Barton and Ivey Jr. (1996) for accelerating the convergence of the algorithm when function evaluations are noisy.

3.2 Benchmark Problems

We have compiled a testbed of 15 problems: 12 were taken from `SimOpt` and the other three (POMDPController, RoutePrices, and TollNetwork) were developed during the course of this project and will be made available on `SimOpt` upon publication. The characteristics of the problems are summarized in Table 1.

Table 1: The benchmark problems and their characteristics: The first column gives the abbreviations used in this paper and hyperlinks to their respective entries on `SimOpt`.

Problems	Name on SimOpt	Dimension	Optimal Solution
Ambulance	Ambulances in a Square	6	Unknown
CtsNews	Continuous Newsvendor	1	Known
DualSourcing	Dual Sourcing	2	Unknown
EOQ	Economic-Order-Quantity	1	Known
FacilityLocation	Facility Location	4	Unknown
MM1	M/M/1 Metamodel	3	Known
MultiModal	A Multimodal Function	2	Known
ParameterEstimation	Parameter Estimation: 2D Gamma	2	Known
POMDPController	Optimal Controller for a POMDP	10	Unknown
ProductionLine	Optimization of a Production Line	3	Unknown
QueueGG1	GI/G/1 Queue	1	Unknown
Rosenbrock	Rosenbrock's Function	40	Known
RoutePrices	Route Prices for Mobility-on-Demand	12	Unknown
SAN	SAN Duration	13	Unknown
TollNetwork	Toll Road Improvements	12	Unknown

4 RESULTS AND DISCUSSION

For every problem, we ran 30 macroreplications of each algorithm. Each macroreplication produced a sequence of estimated best solutions $X(n)$, where n ranges over the simulation budget as specified by the problem's description on `SimOpt`. As a post-processing step, we averaged 30 replications at each solution $X(n)$ to obtain estimates of the objective function $Z(n)$. These post-processing replications were independent of those used to identify the sequence of solutions $X(\cdot)$, and they use common random numbers across all algorithms. We then averaged the 30 estimates of $Z(n)$ to produce the $\bar{Z}(n)$ curve. Since `SPSA` uses the simulation budget as an input, we reran the algorithm with different budgets to produce each point of the $\bar{Z}(n)$ curve. We also calculated 95% normal confidence intervals around $\bar{Z}(n)$. When the results in the $\bar{Z}(n)$ plots require further explanation, we also show plots of the median performance and the first and third quartiles of the 30 (macroreplication) samples of $Z(n)$. We organize our discussion into groupings of problems with similar patterns in their $\bar{Z}(n)$ plots.

CtsNews, MM1, ParameterEstimation, and QueueGG1

All algorithms work well on these low-dimensional problems, quickly converging to good solutions as illustrated in Figure 1 for the problem CtsNews. The plotted quantiles show that the performances of `Nelder-Mead` and `STRONGstg1` are initially more variable than is suggested by the confidence intervals. We also see that `SPSA`'s performance has a large variance for small budgets, as reflected in the width of the confidence intervals. The high variance is not seen in the corresponding quantile plot, suggesting that `SPSA` has occasional very poor performance that shows up only in the below-0.25 quantiles (the worst quarter of the macroreplications).
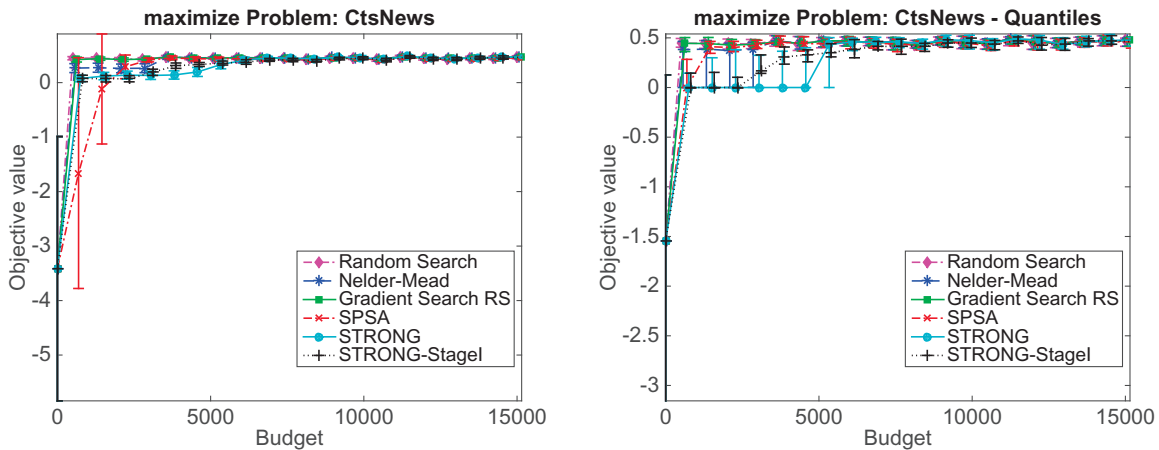
Figure 1: CtsNews: (a) Average Performance, (b) Performance Quantiles.

For the problem MM1, we noticed that the performance of `SPSA` was highly variable and worse than those of the other algorithms, as shown in Figure 2. However, the median performance of `SPSA` is competitive with those of the other algorithms. `SPSA` may be sensitive to the initial solution and occasionally can fail to make progress. This aspect of `SPSA`'s performance appeared in other problems; see the discussion of POMDPController.
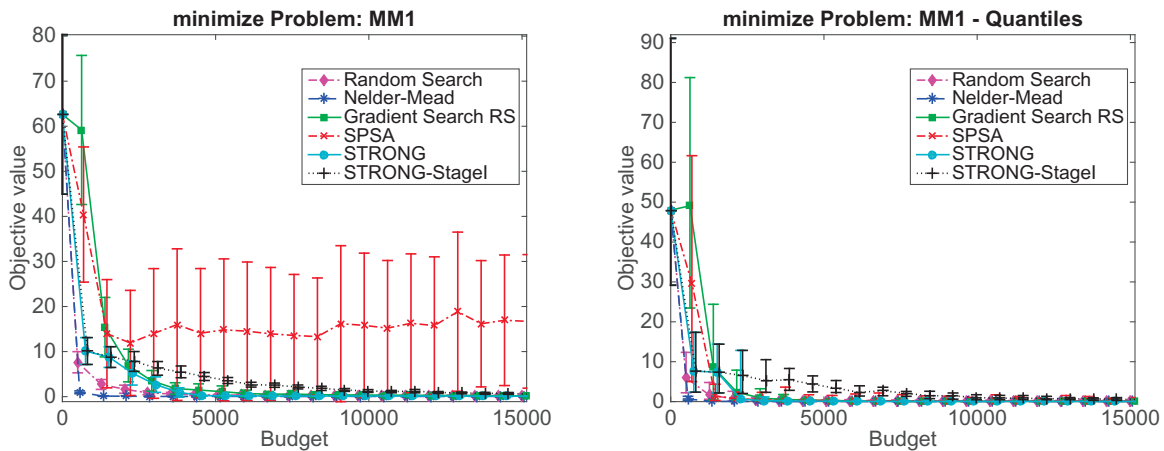

Figure 2: MM1: (a) Average Performance, (b) Performance Quantiles.

EOQ, DualSourcing

Our preliminary results for the problems EOQ and Dual Sourcing indicated that all of the algorithms had similar performance, quickly finding good solutions and then not improving further, as shown in Figure 3. We determined that the generated initial solutions were already near optimal for these two problems, as seen in the vertical scales on those plots. To induce differences in the performances across algorithms, we reran the algorithms, intentionally generating poor initial solutions; see Figures 4 and 5.

In Figure 4, all algorithms except `GradSearch` perform better than `RandomSearch`. `GradSearch` appears to fail due to the shape of the objective function—the function slope is steep to the left of the minimum but very flat to the right. Thus, when starting from a relatively small initial solution, the algorithm can first take a large step to a solution in the flat-slope area. Afterwards, `GradSearch` takes very small
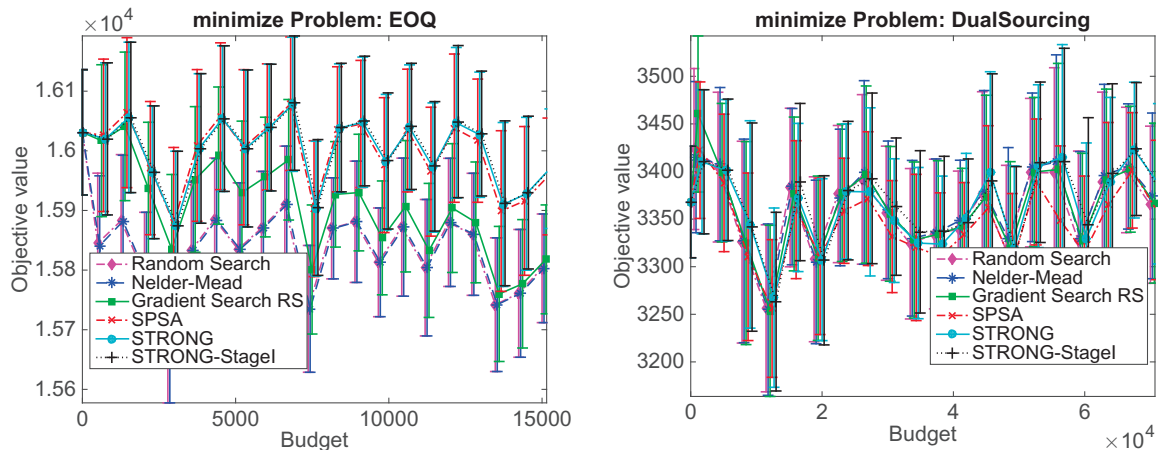
Figure 3: (a) EOQ: Average Performance, (b) DualSourcing: Average Performance.

steps back towards the optimal solution. The performance of `RandomSearch` is highly dependent on the sampling distribution, which in these examples is not well calibrated, leading to poor performance.
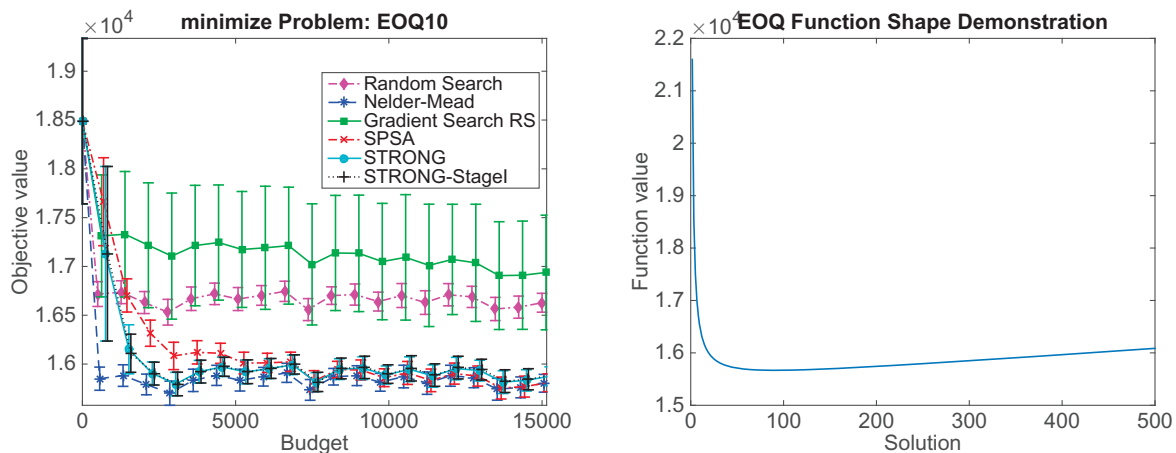

Figure 4: EOQ: (a) Average Performance w/ Bad Initial Solutions, (b) The Objective Function.

For the problem DualSourcing (Figure 5), the extremely bad performance of `RandomSearch` can again be explained by the poorly-claibrated distribution for generating solutions. `SPSA` performs particularly poorly. The relatively weak performance of `Nelder-Mead` on this problem is the result of a few macroreplications on which the algorithm failed to find the optimal solution.

MultiModal, POMDPController, Rosenbrock, Toll Network

For these problems, most algorithms make rapid early progress and then stagnate.

MultiModal is a 2-dimensional problem with 25 widely spaced local optima (Figure 6). Therefore, the local-search algorithms quickly improve on the initial solution, but then fail to make further progress. Meanwhile, the algorithms with restart, namely `RandomSearch` and `GradSearch`, manage to identify better solutions. Compared to `RandomSearch`, `GradSearch` is less efficient because it only employs random restarts once it fails to make progress, so it has fewer opportunities to restart. It only finds the global optimum around half of the time, which helps to explain the high variance of its performance in the results.
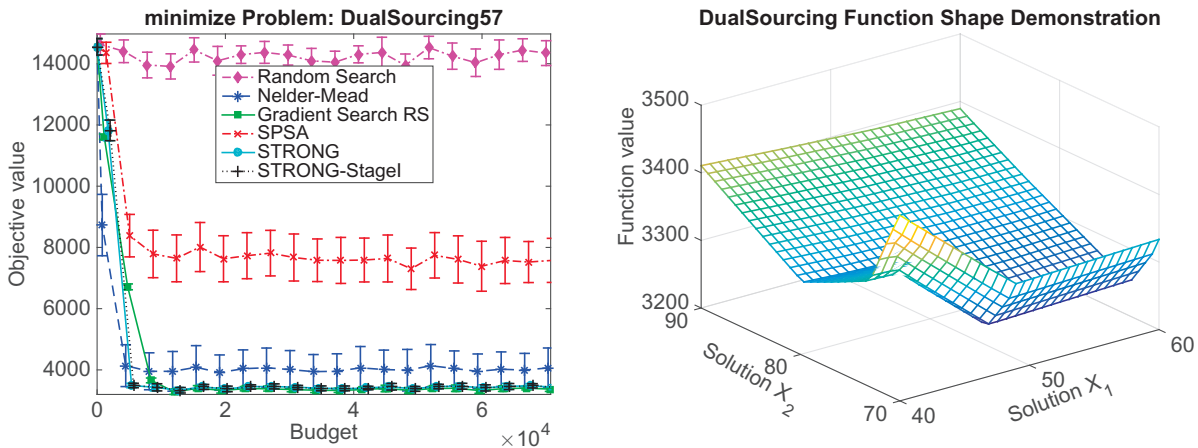
Figure 5: DualSourcing: (a) Average Performance w/ Bad Init., (b) Meshplot of the Objective Function.
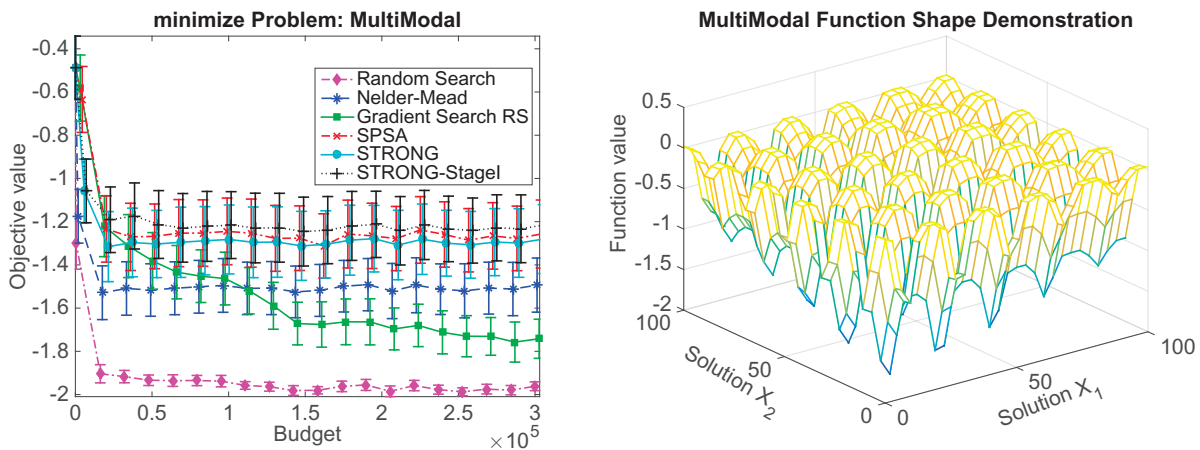

Figure 6: Multimodal: (a) Average Performance, (b) Meshplot of the Objective Function.

We observed that the problem POMDPController was hard for most algorithms to solve (see Figure 7). A plot of the objective function for the case $d = 2$ shows that the objective function is made up of two plateaus with a narrow valley running through one of the plateaus. The valley in which the optimal solution lies is hard to find because it is such a small region in the domain. In higher dimensions, such as the case $d = 10$ that we tested, it is likely that good solutions are located in an even smaller region of the domain. In addition, gradient-based methods struggle because estimates of the gradient on the plateaus are close to zero and therefore dominated by noise. This helps explain why `RandomSearch` outperforms the other methods. The mean performance of `SPSA` is dominated by a few macroreplications on which it performed very poorly.

The Rosenbrock problem has the highest dimension of all our problems. As expected, `RandomSearch` performs poorly in searching a high-dimensional space (Figure 7). `Nelder-Mead` performs exceptionally well. Occasionally, `GradSearch` reaches the boundary of the feasible region and the estimated gradient indicates a search direction outside the boundary. We do not use a projected-gradient algorithm, instead simply pinning solutions back to the boundary, so `GradSearch` can stall on this problem. The two trust-region algorithms appear to struggle, perhaps for the same reason that `GradSearch` struggles. As expected, `SPSA` performs very well in this high-dimensional problem, owing to its cheap gradient estimates.
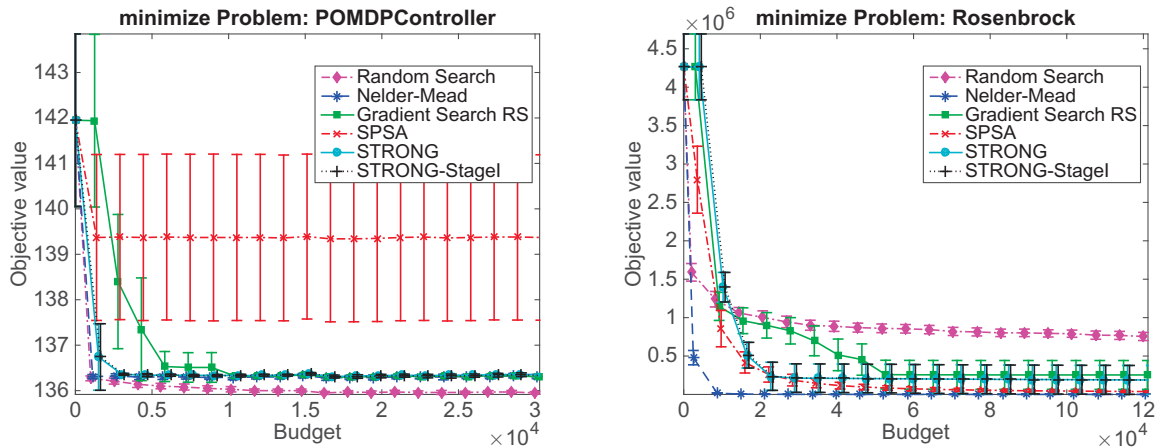
Figure 7: (a) POMDPController: Average Performance, (b) Rosenbrock: Average Performance.

For the TollNetwork problem, `GradSearch` struggled relative to the other algorithms (Figure 8). We do not have an explanation for why `GradSearch` performs so poorly on this problem.
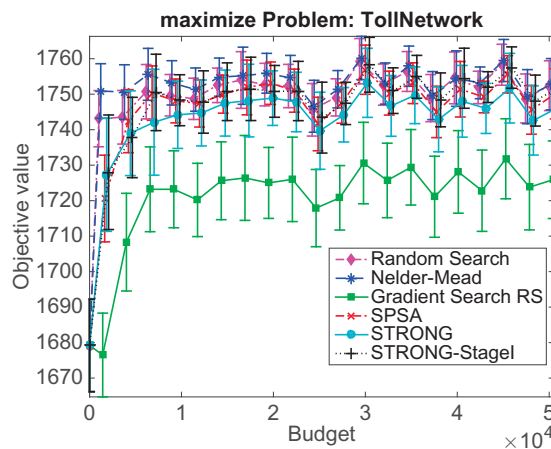

Figure 8: TollNetwork: Average Performance

Ambulance, FacilityLocation, ProductionLine, RoutePrices, SAN

The algorithms have highly varied performance on these five problems.

The problems Ambulance and FacilityLocation have similar problem structure, and the plots exhibit similar trends (Figure 9). `Nelder-Mead` is the stand-out performer, and interestingly `RandomSearch` performs almost as well. `STRONG` probably struggles due to the computational expense of Stage II iterations.

For the problem ProductionLine, all algorithms perform much as they did on the Ambulance problem, with the exception of `SPSA` which is excluded from Figure 10 due to numerical problems in some macroreplications. The RoutePrices problem has dimension $d = 12$, and accordingly `RandomSearch` performs poorly. `Nelder-Mead`, `STRONG` and `STRONGstg1` all perform well on this problem. Of these three algorithms, `Nelder-Mead` is the most consistent performer across macroreplications as suggested by the narrow confidence intervals, but both trust region methods appear to identify a slightly better objective function values for the maximal budget. We do not have an explanation for the poor performance of `GradSearch` and `SPSA` on this problem.
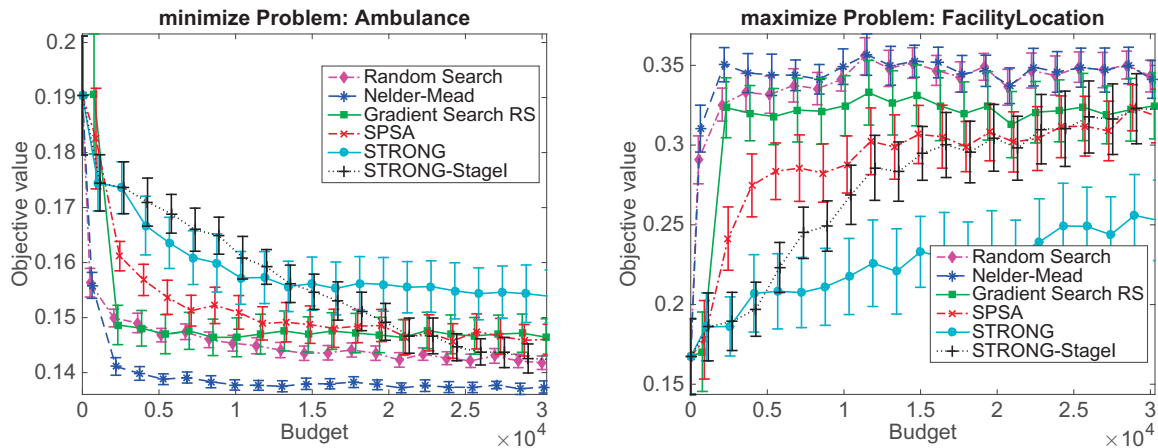
Figure 9: (a) Ambulance: Average Performance, (b) FacilityLocation: Average Performance.
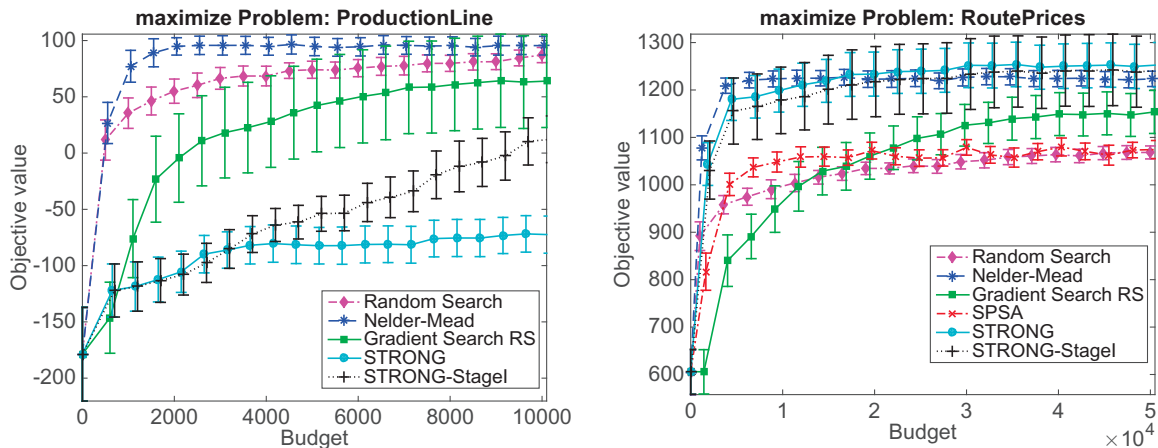

Figure 10: (a) ProductionLine: Average Performance, (b) RoutePrices: Average Performance.

For the problem SAN (Figure 11), the strongest performers are `Nelder-Mead` and `STRONGstg1`. This problem is convex, so we would expect all algorithms to do reasonably well. Nevertheless, `STRONG` struggles, perhaps due to the computational effort in Stage II. `GradSearch` also struggles, perhaps because its test for when to perform a restart is too lenient, leading to many unproductive restarts. `SPSA` performs particularly poorly on this problem.

5 CONCLUSIONS

Perhaps the most important observation in our results is the robust performance of `Nelder-Mead` across almost all problems. It is highly deserving of further study. Interestingly, `STRONGstg1` either compares similarly to, or outperforms, `STRONG`, calling into question the computationally expensive Stage II of `STRONG`. `RandomSearch` performed better than expected, though its performance suffers on higher-dimensional problems or on problems where the sampling distribution is poorly calibrated. `SPSA` performed very well on our highest-dimensional problem (in 40 dimensions), but for most problems it struggled relative to the other algorithms, including problems in dimensions 10–13. Our implementation of `GradSearch` performed moderately well, especially with its use of random restarts and may be worth further development.
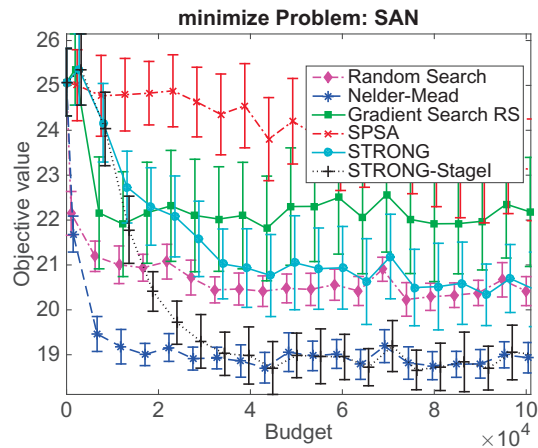
Figure 11: SAN: Average Performance

More generally, our sense is that low-dimensional problems are over-represented in the SimOpt library. It is certainly conceivable that our observations above could change for higher-dimensional problems. The number of problems was small enough that we could present detailed results for a single problem at a time, leading to several insights. If we were to study many more problems, then this presentation would be too cumbersome and a more succinct approach would be necessary. To that end, further research on how to adapt performance profiles to simulation optimization is needed.

In this study we only tackled continuous-variable problems that are either unconstrained or box constrained (with upper and lower bounds on the variables). In future research, we would like to compare algorithms designed for discrete or integer-ordered variables, like COMPASS (Hong and Nelson 2006) and R-SPLINE (Wang, Pasupathy, and Schmeiser 2013).

ACKNOWLEDGMENTS

REFERENCES

Ali, M. M., C. Khompatraporn, and Z. B. Zabinsky. 2005. "A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems". *Journal of Global Optimization* 31 (4): 635–672.

Amaran, S., N. V. Sahinidis, B. Sharda, and S. J. Bury. 2016. "Simulation optimization: a review of algorithms and applications". *Annals of Operations Research* 240 (1): 351–380.

Barton, R. R., and J. S. Ivey Jr.. 1996. "Nelder-Mead simplex modifications for simulation optimization". *Management Science* 42 (7): 954–973.

Bitbucket 2017. "FiniteTimeSimOpt". https://bitbucket.org/poloczek/finitetimesimopt.

Chang, K.-H., L. J. Hong, and H. Wan. 2013. "Stochastic trust-region response-surface method (STRONG)—A new response-surface framework for simulation optimization". *INFORMS Journal on Computing* 25 (2): 230–243.

Dolan, E. D., and J. J. Moré. 2002. "Benchmarking optimization software with performance profiles". *Mathematical Programming Series A* 91 (2): 201–213.

Hong, J. L., and B. L. Nelson. 2006. "Discrete optimization via simulation using COMPASS". *Operations Research* 54 (1): 115–129.

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

Pasupathy, R., and S. G. Henderson. 2006. "A testbed of simulation-optimization problems". In *Proceedings of the 2006 Winter Simulation Conference*, edited by L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 255–263. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pasupathy, R., and S. G. Henderson. 2011. "SimOpt: A library of simulation optimization problems". In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, 4075–4085. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Rios, L. M., and N. V. Sahinidis. 2013. "Derivative-free optimization: A review of algorithms and comparison of software implementations". *Journal of Global Optimization* 56 (3): 1247–1293.

SimOpt Library 2011. "Simulation Optimization (SimOpt) Library". http://www.simopt.org.

Spall, J. C. 1992. "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation". *IEEE Transactions on Automatic Control* 37 (3): 332–341.

Spall, J. C. 1998. "Implementation of simultaneous perturbation algorithm for stochastic optimization". *IEEE Transactions on Aerospace and Electronic Systems* 34 (3): 817–823.

Spall, J. C. 2001. "Simultaneous Perturbation Stochastic Approximation (SPSA)". http://www.jhuapl.edu/SPSA/.

Wang, H., R. Pasupathy, and B. W. Schmeiser. 2013. "Integer-ordered simulation optimization using R-SPLINE: Retrospective search with piecewise-linear interpolation and neighborhood enumeration". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 23 (3): 17:1–24.

AUTHOR BIOGRAPHIES

NAIJIA (ANNA) DONG is a senior student in the School of Operations Research and Information Engineering at Cornell University. Her research interests include simulation optimization, networks, queueing theory and data mining. Her email address is nd2562@columbia.edu.

DAVID J. ECKMAN is a Ph.D. candidate in the School of Operations Research and Information Engineering at Cornell University. His research interests deal with simulation optimization with a focus on ranking-and-selection and exploratory multi-armed bandits. His email address is dje88@cornell.edu.

MATTHIAS POLOCZEK is an assistant professor in the Department of Systems and Industrial Engineering at the University of Arizona. His research focuses on algorithms with performance guarantees in optimization under uncertainty and combinatorial optimization, in particular for applications in simulation optimization, machine learning, and engineering. This research was done while he visited Cornell. His email address is poloczek@email.arizona.edu.

XUEQI ZHAO has graduated from Cornell's ORIE MEng program in December 2016 and will begin her doctoral study in the Department of Industrial and Manufacturing Engineering at Penn State University from September 2017. Her research interests focus on data science and simulation optimization algorithms. Her email address is xuz206@psu.edu.

SHANE G. HENDERSON is a professor in the School of Operations Research and Information Engineering at Cornell University. His research interests include discrete-event simulation and simulation optimization, and he has worked for some time with emergency services and bike sharing applications. He co-edited the Proceedings of the 2007 Winter Simulation Conference. His web page is http://people.orie.cornell.edu/~shane.