SAND2011-0723C

Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 00 (2011) 1–10
Procedia Computer Science

International Conference on Computational Science, ICCS 2010

An Asychronous Parallel Hybrid Approach for MINLPs

K.R. Fowler[a], T. Kopp[a], J. Orsini[a], J.D. Griffin[b], G.A. Gray[c,1,*]

[a] *Department of Mathematics and Computer Science, Clarkson University*
[b] *SAS*
[c] *Sandia National Labs*

Abstract

To address simulation-based mixed-integer problems, a hybrid algorithm was recently proposed that combines the global search strengths and the natural capability of a genetic algorithm to handle integer variables with a local search on the real variables using an implementation of the generating set search method [1]. Since optimization is guided only by function values, the hybrid is designed to run asynchronously on a parallel platform. The algorithm has already been shown to perform well on a variety of test problems, and this work is a first step in understanding how the parallelism and local search components influence the search phase of the algorithm. We show that the hybridization can improve the capabilities of the genetic algorithm by using less function evaluations to locate the solution and provide a speed-up analysis on a standard mixed-integer test problem with equality and inequality constraints.

Keywords: genetic algorithm, pattern search, asynchronous, mixed-integer nonlinear programming

1. Introduction

The need for reliable and efficient optimization algorithms that do not require derivatives is common across engineering disciplines. In general, the optimal design process requires such algorithms to work in conjunction with simulation tools, resulting in what is known as black-box optimization. For example, the simulation may require the solution to a system of partial differential equations that describe a physical phenomena. These problems are challenging in that optimization must be guided by objective function (and possibly constraint) values that rely on a computer simulation, without any additional knowledge other than the output from the simulation itself. The simulation may be computationally expensive and add undesirable features to the underlying problem such as low amplitude noise, discontinuities, or hidden constraints (i.e. when the program simply fails to return a value due to its own internal solver failure). Derivative-free optimization (DFO) methods have been developed, analyzed, and demonstrated successfully over the last several decades on a wide range of applications [2]. Because DFO methods only rely on function values, parallelism is often straightforward and, in the case of expensive simulation calls, can make otherwise intractable problems solvable.

Hybrid DFO algorithms have emerged to overcome inherent weaknesses and exploit strengths of the methods being paired [3, 4, 5]. Often, the hybrid algorithms are designed to address problems that could not otherwise be

* *Email address:* gagray@sandia.gov (G.A. Gray)
[1] Corresponding author

solved. In this work, we focus on the parallelism of a hybrid evolutionary algorithm with a local search that was designed for simulation-based mixed-integer problems with nonlinear constraints [1]. The performance of the hybrid was demonstrated on a suite of standard test problems and on two applications from hydrology ([6, 7, 1]that were known to be challenging for a wide range of DFO methods [8, 9]. Some of those challenges, which are not unique to environmental engineering, included discontinuous optimization landscapes, low amplitude noise, multiple local minima. Specifically, in [8], a comparison of derivative-free methods on the hydrology applications showed that a genetic algorithm (GA) performed well in terms of identifying the correct integer variables but then failed to achieve sufficient accuracy for the real variables. On the other hand, given a reasonable initial iterate with respect to the integer variables, the local search methods showed fast convergence. These observations motivated the pairing of the GA with a generating set search approach, referred to as **E**volutionary **A**lgorithms **G**uiding **L**ocal **S**earch (EAGLS). The resulting algorithm pairs the binary mapping of the genetic algorithm to handle integer variables with asynchronous, parallel local searches on only the real variables. The resulting method has strong global search aspects and can still maintain high accuracy from the local search phase.

Previous studies focused on the ability of EAGLS to solve a variety of MINLPs with varying difficulties in constraint formulations and problem size. In [1], EAGLS was able to solve a water supply hydrology application that previously could not be solved without significant parameter tuning of either of the two software tools that were merged to create the hybrid. Little work has been done to understand how the asynchronous parallelism that is inherent in the implementation impacts the search phase of the algorithm. This work is a first attempt at using parallel performance measures to understand the algorithms strengths and weaknesses.

For this work, we consider objective functions of the form $f : \mathbb{R}^{n_r+n_z} \to \mathbb{R}$ and mixed-integer nonlinear optimization problems of the form

$$\min_{p\in\Omega} f(p). \tag{1}$$

Here n_r and n_z denote the number of real and integer variables and $x \in \mathbb{R}^{n_r}, z \in \mathbb{Z}^{n_z}$. In practice, Ω may be comprised of component-wise bound constraints on the decision variable in combination with linear and nonlinear equality or inequality constraints. Often, Ω may be further defined in terms of state variables determined by simulation output. We proceed by first reviewing the genetic algorithm, the generating set search method, and software that are hybridized to form the new algorithm. We then present numerical results and outline future directions.

2. EAGLS

2.1. Genetic Algorithms

The EAGLS approach combines a genetic algorithm and a generating set search approach. GAs [10, 11, 12] are one of the most widely-used DFO methods and are part of a larger class of evolutionary algorithms called population-based, global search, heuristic methods [10]. GAs are based on biological processes such as survival of the fittest, natural selection, inheritance, mutation, or reproduction. Design points are coded as "individuals" or "chromosomes", typically as binary strings, in a population and then undergo the above operations to evolve towards a better fitness (objective function value).

A simple GA can be outlined with:

1. Generate a random/seeded initial population of size n_p
2. Evaluate the fitness of individuals in initial population
3. Iterate through the specified number of generations:
   (a) Rank fitness of individuals
   (b) Perform selection
   (c) Perform crossover and mutation
   (d) Evaluate fitness of newly-generated individuals
   (e) Replace non-elite members of population with new individuals

During the selection phase, better fit individuals are arranged randomly to form a mating pool on which further operations are performed. Crossover attempts to exchange information between two design points to produce a new point that preserves the best features of both 'parent points'. Mutation is used to promote a global search and prevent

stagnation at a local minimum. Termination of the algorithm is typically based on a function evaluation budget that is exhausted as the population evolves through generations.

Often, GAs are criticized for their computational complexity and dependence on optimization parameter settings, which are not known a priori [13, 14, 15]. Parameters like the population size, number of generations, as well as the probabilities and distribution indices chosen for the crossover and mutation operators affect the performance of a GA [16, 17]. Also, since the GA incorporates a randomness to the search phase, multiple optimizations are often useful to exhaustively search the design space. However, if the user is willing to spend a large number of function evaluations, a GA can help provide insight into the design space and locate initial points for fast, local, single search methods. The GA has many alternate forms and has been applied to a wide range of engineering design as shown in references such as [18]. Moreover, hybrid GAs have been developed at all levels of the algorithm and with a variety of other global and local search DFO methods. See for example [19, 3, 4] and the references therein.

The EAGLS software package was created using the Non-dominated Sorting Genetic Algorithm (NSGA-II) software, which is described in [20, 21, 22, 23]. Although a variety of genetic algorithms exist, the NSGA-II has been applied to both single and multi-objective problems for a wide range of applications and is well supported. In particular, it is deigned to be used "off-the-shelf" which made it a good candidate for hybridization.

2.2. Generating Set Search and APPS

Asynchronous Parallel Pattern Search (APPS) [24, 25] is a direct search methods which uses a predetermined pattern of points to sample a given function domain. APPS is an example of a generating set search (GSS), a class of algorithms for bound and linearly constrained optimization that obtain conforming search directions from generators of local tangent cones [26, 27]. In its simplest form, the method evaluates the objective function on a stencil of points and if a better point is found, the stencil is moved to that point, otherwise the size of the stencil is reduced. Optimization is terminated either based on a function evaluation budget or when the stencil becomes sufficiently small. The basic GSS algorithm is:

Let x_0 be the starting point, Δ_0 be the initial step size, and $\mathcal{D} = \{d_i\}_{i=1}^{2n_r}$ be the set of positive spanning directions.

While *not converged* Do

1. Generate trial points $Q_k = \{x_i + \tilde{\Delta}_k d_i \Big| \ 1 \le i \le |\mathcal{D}|\}$ where $\tilde{\Delta}_k \in [0, \Delta_k]$ denotes the maximum feasible step along d_i.
2. Evaluate trial points (possibly in parallel)
3. If $\exists\ x_q \in Q_k$ such that $f(x_q) - f(x_k) < \alpha\Delta_k^2$
   Then $x_{k+1} = x_q$ (successful iteration)
   Else $x_{k+1} = x_k$ (unsuccessful iteration) and $\Delta_{k+1} = \Delta_k/2$ (step size reduction)

The majority of the computational cost of pattern search methods is the $2n_r$ function evaluations, so parallel pattern search (PPS) techniques have been developed to perform function evaluations simultaneously on different processors [28, 29]. For example, for a simple two-dimensional function, consider the illustrations in Figure 1 taken from [30]. First, the points f, g, h, and i in the stencil around point c are evaluated. Then, since f results in the smallest function value, the second picture shows a new stencil around point f. Finally, in the third picture, since none of the iterates in this new stencil result in a new local minima, the step size of the stencil is reduced.

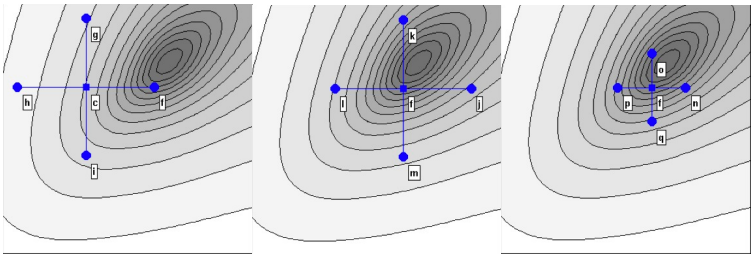
Note that in a basic GSS, after a *successful* iteration (one in which a new best point has been found), the step size is either left unchanged or increased. In contrast, when the iteration was unsuccessful, the step size is necessarily reduced. A defining difference between the basic GSS and APPS is that the APPS algorithm processes the directions independently, and each direction may have its own corresponding step size. Global convergence to locally optimal points is ensured using a sufficient decrease criteria for accepting new best points. A trial point $x_k + \Delta d_i$ is considered better than the current best x_k point if

$$f(x_k + \Delta d_i) - f(x_k) < \alpha\Delta^2, \tag{2}$$

for $\alpha > 0$.

Because APPS processes search direction independently, it is possible that the current best point is improved before all the function evaluations associated with a set of trial points Q_k have been completed. These results are

Figure 1: Illustration of the steps of Parallel Pattern Search (PPS) for a simple two-dimensional function. On the left, an initial PPS stencil around starting point c is shown. In the middle, a new stencil is created after successfully finding a new local min (f). On the left, PPS shrinks the stencil after failing to find a new minimum.



(a) Initial (b) Success (c) Failure

referred to as *orphaned points* as they are no longer tied to the current search pattern and attention must be paid to ensure that the sufficient decrease criteria is applied appropriately. The support of these orphan points is a feature of the APPS algorithm which makes it naturally amenable to a hybrid optimization structure. Iterates generated by alternative algorithms can be simply be treated as orphans without the loss of favorable theoretical properties or local convergence theory of APPS.

2.3. Why EAGLS works

EAGLS combines the NSGA-II with the APPSPACK software [31]. APPSPACK is written in C++ and uses MPI [32, 33] for parallelism. Function evaluations are performed through system calls to an external executable which can written in any computer language. This simplifies its execution and also makes it a good candidate for inclusion in a hybrid scheme. Moreover, it should be noted that the most recent version of APPSPACK can handle linear constraints [27, 34], while a software package called HOPSPACK builds on the APPSPACK software and includes a GSS solver that can handle nonlinear constraints [35, 36]. To implement EAGLS, as in [37], a preliminary version of HOPSPACK was used.

The EAGLS algorithm is designed to exploit parallelism. A goal of a parallel program is to ensure that all available processors are continuously being used. However, in practice this is often not the case. To understand this more fully in our context, consider a hypothetical black-box bound constrained optimization problem that has two real variables and an objective function with an evaluation time of at least one hour; further, we assume the user has 128 nodes with 2 processors each. There are a number of advantages that come from the use of parallelism in this context.

- Most local search algorithms (even asynchronous parallel ones) have a cap on the maximum number of processors they can effectively use. For our example problem, APPS will generate at most 4 trial-point per iteration. For the first hour APPS is called 252 processors will be idle. The user would need to start by hand 64 different instances of APPS centered at unique starting points, to fully exploit the computational power at hand with APPS alone.

- Most algorithms are synchronous by design, and parallel versions typically run in a "batch" mode. For example, a genetic algorithm requires all points in the current generation be evaluated before creating the next. Suppose a parallel GA uses a population of size 256 and submits all 256 points to be evaluated in parallel. Before the second iteration can begin, all 256 points must be evaluated; if all evaluations are complete but one, then the entire optimization processes is halted until this final evaluation is completed, even if this remaining evaluation takes hours longer to complete. Thus synchronous parallel algorithms necessarily move at the rate of the slowest evaluation.

The downside described in the preceding bullets are actually advantageous for hybrid algorithms. Rather than attempt to redesign APPS so that it will submit more points each iteration or invent a new asynchronous genetic algorithm that seeks to update multiple generations asynchronously, we simply tie multiple algorithms together loosely, pooling the resources in such a way that any unused resources can be shared. In the case of EAGLS, a single GA is run, and

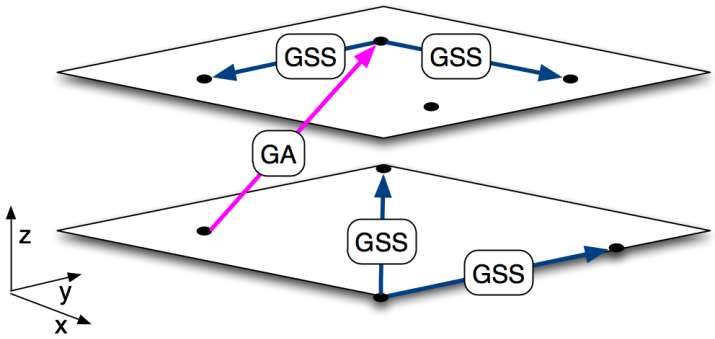


Figure 2: In EAGLS the genetic algorithm optimizes over both integers and real variables, while local search instances work solely within a given integer plane [1].

remaining idle processors are used to perform local searches. However, because local searches are often much faster then a GA at finding a local minimum, priority is given each generation to the local searches in the evaluation queue until that iterations local search evaluation budget has been expended. An immediate consequence and benefit of the EAGLS structure is that there is virtually no cap on the maximum number of processors that can be utilized for a given problem. At the same time, even with a few extra processors, significant wall-clock gains can be achieved, as the local search can be used to quickly find the global minimum once the GA is sufficiently near.

2.4. EAGLS Algorithm

EAGLS uses the GA's handling of integer and real variables for the global search, and APPS's handling of real variables in parallel for local search. Note that a MINLP could be immediately reduced to an integer programming problem if there was an analytic formula that provided x^* where

$$x^* = \arg\min_x f(x, z)$$

given an integer variable z. Though for a general MINLP, such a formula may not exist, local searches can be used (in parallel) to repeatedly replace (x, z) pairs in the GA population pool with $(\hat{x}, z)$, where $\hat{x}$ is an improved estimate of x^* provided by a local search. The GA still governs point survival, mutation, and merging as an outer iteration, but, during an inner iteration, individual points are improved via APPS applied to the real variables, with the integer variables held fixed. For simplicity, consider the parallel synchronous EAGLS algorithm with k local searches:

1. Evaluate initial population in parallel
2. While *not converged* Do
   (a) Choose a subset $\mathcal{L}$ of k points from current population for local search
   (b) Simultaneously run k instances of APPS centered at points in $\mathcal{L}$
   (c) Replace respective points with their optimized values
   (d) Perform selection, mutation, crossover
   (e) Evaluate new GA points in parallel

To select points for the local search, EAGLS uses a ranking approach that takes into account individual proximity to other, better points (see Figure 2). The goal of this step is to choose promising individuals representing distinct integer subdomains. The EAGLS algorithm allows the local search and the GA to run simultaneously using the same pool of evaluation processors. For the most part, the GA and each local search run asynchronously. However, after each GA generation, a new batch of local searches are created and given priority in the evaluation queue. This implies that given a adequate number of local search instances, the GA generations and local search generation will necessarily be nested, as the number of local search trial-points will always be greater than the number of available processors in the evaluation queue. This forces the GA to wait until the local search generation depletes its current evaluation budget prior to proceeding. Once the GA population has been evaluated, the local searches begin and operate asynchronously.

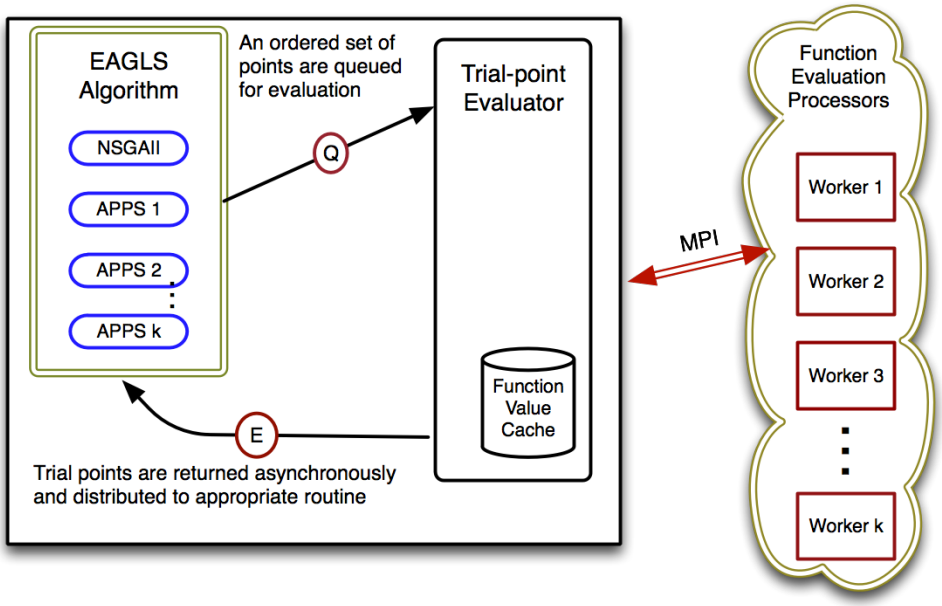


Figure 3: The EAGLS user can decide the population size and the number of local searches in an input file. The algorithms are run asynchronously in parallel with the local searches periodically inserting new improved points into the current GA population.

To avoid re-evaluating points, all function values are stored in cache. The external parallel paradigm is nearly identical to that used in [37, 38]. Whenever an improved point is found with respect to the real variables, the corresponding population member is immediately updated. See Figure 3 for a short point-flow sketch of this process.

3. Numerical Results

3.1. Test Problem

To evaluate the parallelism of EAGLS we consider two studies. In the first, we fix all the optimization algorithm parameters and increase the number of processors used. In the second, we fix the number of processors to 16 and vary only the number of local searches while all other optimization parameters are held fixed. We use a standard mixed-integer test problem taken from [39]. The decision variables are $p = (z_1, z_2, z_3, x_1, x_2)^T$ with bound constraints given by

$$p \in \Omega = \{p | z_1, z_2, z_3 \in \{0, 1\}, x_1, x_2 \in [0, 10]\}.$$

We seek to minimize the objective function $f(p)$ where

$$f(p) = 2x_1 + 3x_2 + 1.5z_1 + 2z_2 - 0.5z_3 \tag{3}$$

subject to the following constraints,

$$\begin{aligned} c_1(p) &= x_1^2 + z_1 - 1.25 = 0 \\ c_2(p) &= x_2^{1.5} + 1.5z_2 - 3.00 = 0 \\ c_3(p) &= x_1 + z_1 - 1.60 \leq 0 \\ c_4(p) &= 1.333x_2 + z_2 - 3.00 \leq 0 \\ c_5(p) &= -z_1 - z_2 + z_3 \leq 0. \end{aligned} \tag{4}$$

The constraints on both the integer and real variables make this problem challenging. For constraint handling, we use the ℓ_1 and the ℓ_1-smoothed penalty function where the constraint violation is incorporated with the objective function to form a corresponding merit function [37]. The known solution has a function value of 7.667, and there

Table 1: Optimization Parameters

Parameter	Value
Population size	40
Number of Generations	250
Real Crossover Probability	0.9
Real Mutation Probability	0.5
Binary Crossover Probability	0.9
Binary Mutation Probability	0.0125
GSS Contraction Factor	0.5
GSS Sufficient Decrease Factor	1e-9
GSS Step Tolerance	1e-5
Maximum Generation Evaluations	840
Maximum Function Evaluations	3000

is a local minimum with function value 7.931. To add computational expense to each function evaluation and test the asynchronous nature of the algorithm, we add a random pause between one and three seconds to each function evaluation. This approach was used to test parallel optimization approaches in [24, 37]

3.2. Algorithmic Parameters and Platform

Since the solution to test problem is known, we stop when the best point found is within 1% of the known solution. We provide the other relevant optimization parameters in Table 1. The numerical experiments were performed on a 102 processor Beowulf blade cluster (IBM e1350) with 3.0 Ghz Intel Xeon processors and Myrinet Networking.

3.3. Varying Number of Processors

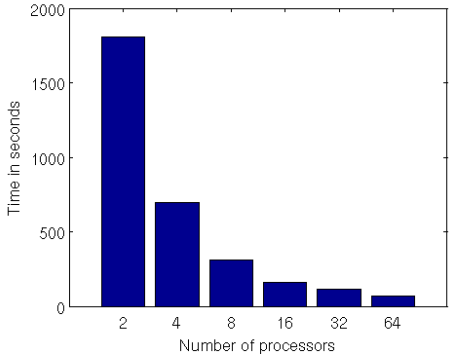
Since the GA has stochastic optimization parameters and APPS is asynchronous, EAGLS is not a deterministic method, thus each optimization experiment was run five times and average values are reported. This approach has been used in numerous studies for APPS [37]. Average run times and number of function evaluations required for convergence are shown in Figure 4 as the number of processors doubles from 2 to 64. For these experiments EAGLS used 8 local searches. Since there are only $n_r = 2$ real variables, for each local search APPSPACK would not see increased speed up beyond $2n_r = 4$ processors for a total of 32 while the additional processors can be used to evaluate the GA population. The figure on the left shows the speed-up one would expect. The figure on the right is interesting in that the number of function evaluations increases with the number of processors. This is because as APPSPACK is run on more processors, the algorithm may move the stencil to a new location if a point is found with a lower function value but older points are not deleted from the queue if sufficient processors are allocated. So if a point from an older stencil does return a lower function value, the algorithm would move back to that location and continue. Note that because significantly more processors are being used, the computational time still shows linear speed-up despite the increased number of function evaluations.

3.4. Varying Number of Local Searches

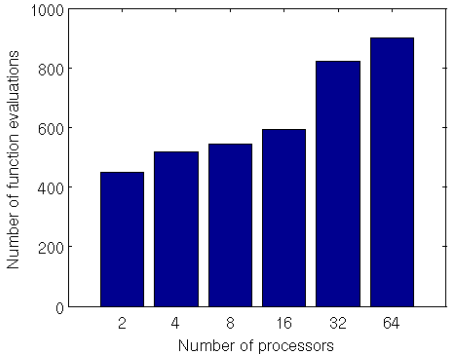
To further understand how the asynchronous nature of APPSPACK impacts the search phase of EAGLS, we vary the number of local searches. For these experiments, 16 processors were used and all optimization algorithmic parameters were fixed except the number of local searches, which was varied from 4 to 8. We also consider the case of no local searches, which means EAGLS is simply a genetic algorithm with function evaluations performed in parallel. Figure 5 shows the average run times and number of function evaluations needed for convergence.

The local searches have a significant impact on the optimization history using roughly one fifth of the computational effort of the GA alone. As the number of local searches increases, the number of function evaluations increases as one would expect but it is not significant. This is due in part to the fact that the algorithm is terminating based on proximity to a known solution. Future work will include exploring the behavior on larger dimensional problems which may show more dynamic results in terms of the optimal number of local searches, but for this work we are staying in the context of simulation-based MINLPs which typically are not too large. We should further note that

Figure 4:

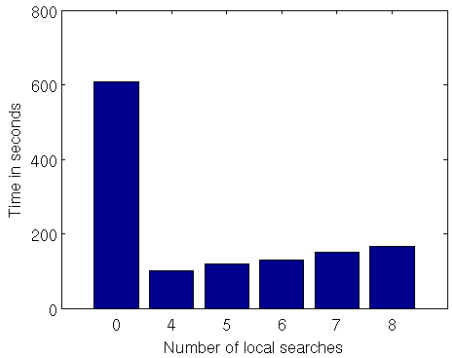


(a) Run times

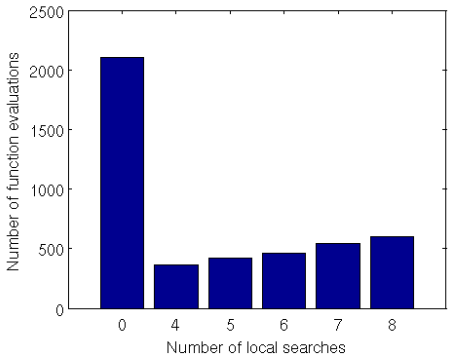


(b) Number Function Evaluations

Figure 5:



(a) Run times



(b) Number Function Evaluations

this test problem does have a feasible local minimum with a function value of roughly 7.931, and EAGLS avoided convergence to this suboptimal point in all trials.

4. Conclusions

These experiments are the first step in understanding an asynchronous hybridization of a genetic algorithm with a local search based on a generating set search method for mixed-integer problems. This approach has extended the APPSPACK software to handle integer variables, improved its global search capabilities, and added parallelism and a local search to the NSGA-II software package. The tests done here are promising in showing that using local searches can help accelerate the convergence of the GA but also indicate that there is a complex interaction among algorithm parameters. The GA is well-known to be sensitive to parameter settings and the addition of an asynchronous local search with additional parameters warrants a more extensive study to better guide users. Future work will include a sensitivity study similar to that in [40] to understand the interaction and main effects of the optimization settings.

[1] J. Griffin, K. Fowler, G. Gray, T. Hemker, M. Parno, Derivative-free optimization via evolutinary algorithms guiding local search (eagls) for minlp, accepted to Pacific Journal of Optimization.
[2] A. Conn, K. Scheinberg, L. N. Vincente, Introduction to Derivative-Free Optimization, SIAM, Philadelphia, PA, 2009.
[3] E. Talbi, A taxonomy of hybrid metaheurtistics, J. Heuristics 8 (2004) 541–564.
[4] G. R. Raidl, A unified view on hybrid metaheuristics, in: HM06: Third International Workshop on Hybrid Metaheuristics, 2006, pp. 1–12.
[5] E. Alba, Parallel Metaheuristics, John Wiley & Sons, Inc., 2005.
[6] G. A. Gray, K. R. Fowler, J. D. Griffin, A hybrid optimization scheme for solving the hydraulic capture problem with an unknown number of wells, in: Conference on Soft Computing, 2009.
[7] G. A. Gray, K. Fowler, J. D. Griffin, Hybrid optimization schemes for simulation based problems, Procedia Comp. Sci. 1 (1) (2010) 1343–1351.
[8] K. R. Fowler, et al., A comparison of derivative-free optimization methods for water supply and hydraulic capture community problems, Adv. Water Resourc. 31 (5) (2008) 743–757.
[9] K. R. Fowler, C. T. Kelley, C. T. Miller, C. E. Kees, R. W. Darwin, J. P. Reese, M. W. Farthing, M. S. C. Reed, Solution of a well-field design problem with implicit filtering, Opt. Eng. 5 (2004) 207–234.
[10] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, 1989.
[11] J. H. Holland, Adaption in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.
[12] J. H. Holland, Genetic algorithms and the optimal allocation of trials, SIAM J. Comput. 2.
[13] K. A. DeJong, W. M. Spears, An analysis of the interacting roles of population size and crossover in genetic algorithms (1990).
[14] J. Grefenstette, Optimization of control parameters for genetic algorithms, IEEE Trans. Sys. Man Cybernetics SMC-16 (1) (1986) 122–128.
[15] F. G. Lobo, C. F. Lima, Z. Michalewicz (Eds.), Parameter settings in evolutionary algorithms, Springer, 2007.
[16] P. Reed, B. Minsker, D. Goldberg, Designing a competent simple genetic algorithm for search and optimization, Water Resources Research 36 (12) (2000) 3757–3761.
[17] A. Mayer, C. Kelley, C. Miller, Optimal design for problems involving flow and transport phenonmena in saturated subsurface systems, Advances in Water Resources 12 (2002) 1233–1256.
[18] C. Karr, L. Freeman, Industrial Applications of Genetic Algorithms, International Series on Computational Intelligence, CRC Press, 1998.
[19] C. Blum, M. B. Aquilera, A. Roli, M. Sampels, Hybrid Metaheuristics, Studies in Computational Intelligence, Springer, 2008.
[20] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multi-objective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.
[21] E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: Empirical results, Evolutionary Computation Journal 8 (2) (2000) 173–195.
[22] K. Deb, An efficient constraint handling method for genetic algorithms, Computer Methods in Applied Mechanics and Engineering 186 (2–4) (2000) 311–338.
[23] K. Deb, T. Goel, Controlled elitist non-dominated sorting genetic algorithms for better convergence, in: E. Zitler, K. Deb, L. Thiele, C. Coello-Coello, D. Corne (Eds.), Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization EMO 2001, Lecture Notes on Computer Science, , 2001, pp. 67–81.
[24] P. D. Hough, T. G. Kolda, V. Torczon, Asynchronous parallel pattern search for nonlinear optimization, SIAM J. Sci. Comput. 23 (2001) 134–156.
[25] T. G. Kolda, Revisiting asynchronous parallel pattern search, Tech. Rep. SAND2004-8055, Sandia National Labs, Livermore, CA 94551 (February 2004).
[26] R. M. Lewis, A. Shepherd, V. Torczon, Implementing generating set search methods for linearly constrained minimization, Tech. Rep. WM-CS-2005-01, Department of Computer Science, College of William & Mary, Williamsburg, VA, revised July 2006. (2005).
[27] T. G. Kolda, R. M. Lewis, V. Torczon, Stationarity results for generating set search for linearly constrained optimization, SIAM J. Optim. 17 (4) (2006) 943–968.
[28] J. E. Dennis, Jr., V. Torczon, Direct search methods on parallel machines, SIAM J. Optim. 1 (1991) 448–474.
[29] V. Torczon, PDS: Direct search methods for unconstrained optimization on either sequential or parallel machines, Tech. Rep. TR92-09, Rice Univ., Dept. Comput. Appl. Math., Houston, TX (1992).
[30] G. Gray, K. Fowler, Computational Optimization and Applications in Engineering and Industry, Springer, 2011, Ch. Traditional and Hybrid Derivative-free Optimization Approaches for Black-box Optimization.

[31] G. A. Gray, T. G. Kolda, Algorithm 856: APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization, ACM TOMS 32 (3) (2006) 485–507.
[32] W. Gropp, E. Lusk, N. Doss, A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, Parallel Comput. 22 (1996) 789–828.
[33] W. D. Gropp, E. Lusk, User's guide for `mpich`, a portable implementation of MPI, Tech. Rep. ANL-96/6, Mathematics and Computer Science Division, Argonne National Lab (1996).
[34] J. D. Griffin, T. G. Kolda, R. M. Lewis, Asynchronous parallel generating set search for linearly-constrained optimization, SIAM J. Sci. Comp. 30 (4) (2008) 1892–1924.
[35] J. D. Griffin, T. G. Kolda, Nonlinearly-constrained optimization using heuristic penalty methods and asynchronous parallel generating set search, Appl. Math. Res. eXpress 2010 (1) (2010) 36–62.
[36] T. D. Plantenga, HOPSPACK 2.0 User Manual (v 2.0.1), Tech. Rep. SAND2009-6265, Sandia National Labs, Livermore, CA (2009).
[37] J. D. Griffin, T. G. Kolda, Asynchronous parallel hybrid optimization combining DIRECT and GSS, Optim. Meth. Software 25 (5) (2010) 797–817.
[38] G. A. Gray, J. D. Griffin, et al., HOPSPACK: Hybrid optimization parallel search package, Tech. Rep. SAND2008-8057, Sandia National Labs, Livermore, CA 94551-0969 (2008).
[39] G. Kocis, I. Grossmann, Global optimization of nonconvex mixed-integer nonlinear programming (minlp) problems in process synthesis, Ind. Eng. Chem. Res. 27 (1988) 1407–1421.
[40] L. Matott, S. Bartlelt-Hunt, A. Rabideau, K. Fowler, Application of heuristic techniques and algorithm tuning to a multilayered sorptive barrier system, Environmental Science & Technology 40 (2006) 6354 – 6360.