Yokohama Publishers
ISSN 1349-8169 ONLINE JOURNAL

Pacific Journal of Optimization
Vol. 7, No. 3, pp. 425–442, September 2011

DERIVATIVE-FREE OPTIMIZATION VIA EVOLUATIONARY ALGORITHMS GUIDING LOCAL SEARCH (EAGLS) FOR MINLP*

J.D. Griffin, K.R. Fowler, G.A. Gray, T. Hemker and M.D. Parno

Abstract: Derivative-free optimization approaches are commonly used for simulation-based design problems when objective function and possibly constraint evaluations have a black-box formulation. A variety of algorithms have been developed over the last several decades to address the inherent challenges such as computationally expensive function evaluations, low amplitude noise, nonsmoothness, nonconvexity, and disconnected feasible regions. Hybrid methods are emerging within the direct search community as new tools to overcome weaknesses while exploiting strengths of several methods working together. In this work, we extend the capabilities of a parallel implementation of the generating set search (GSS) method, which is a fast local derivative-free approach, to handle integer variables. This is achieved with a hybrid approach that uses a genetic algorithm (GA) to handle the integer variables. Promising points are selected as starting points for the GSS local search with the integer variables held fixed before being passed back to the GA for the standard selection, mutation and crossover operations for the next iteration. We provide promising numerical results on three mixed integer problems; one based on the design of a compression spring, a simulation-based problem from hydrology, and a standard problem taken from the literature.

Key words: *mixed-integer problems, generating set-search, genetic algorithm*

Mathematics Subject Classification: *90C11, 90C56, 65K05, 49M30*

1 Introduction

In this paper we explore the asynchronous parallel hybrid combination of an existing heuristic algorithm (NSGA-II) with an existing direct search algorithm (APPSPACK). Heuristic-based algorithms, such as genetic algorithms, are attractive in that they often naturally support discrete variable manipulation; however they can require an inordinate number of function evaluation to converge to a local minimum. Generating Set Search (GSS) is a derivative-free optimization method where minimization is guided only by function values evaluated using a positive-spanning basis [31, 32]. It is well-suited for simulation-based optimization problems where function evaluations are expensive, nonsmooth, and possibly discontinuous. Even if they ultimately approximate a smooth function, the objective and constraint derivatives are typically unknown and numerical approximations may be unreliable due to noise. GSS is easily parallelized, and can be made to run asynchronously to

*This work was partially supported by the American Institute of Mathematics. The work of Gray was supported by Sandia National Laboratories, a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000. Thanks to I. Akrotirianakis for his input and to R.M. Fowler for assistance with the figures.

Copyright © 2011 Yokohama Publishers http://www.ybook.co.jp

minimize load imbalance. Because GSS algorithms are only loosely coupled with derivatives, they are less likely to become stuck at nearby local minima when compared to derivative-based algorithm; however, they are nonetheless a local search algorithm and cannot move on to explore alternate regions once all the existing trial-points lie within a single basin (or locally convex region). Further, GSS algorithms do not natively support categorical variables or mixed-integer problems.

Abramson et al. [1] recently extend a direct search (MADS) algorithm to handle mixed-variables by incorporating user-defined neighborhoods for the categorical variables. In this paper, we explore the use of a genetic algorithm in place of user-defined neighborhoods. This is done using a loosely coupled parallel hybrid optimization framework where multiple instances of direct search algorithms are run collaboratively in parallel for each genetica algorithm generation. Because we assume that the discrete variables are non-relaxable, the direct search instance each optimize only with respect to the continuous variables, holding the integer (or categorical) variables fixed temporarily. All evaluations are computed asynchronously in parallel.

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

$$\begin{array}{ll} \underset{p}{\text{minimize}} & f(p) \\ \text{subject to} & c(p) \leq 0 \\ & p_\ell \leq p \leq p_u, \end{array} \tag{1.1}$$

where

$$p_\ell = \begin{pmatrix} z_\ell \\ x_\ell \end{pmatrix},\ p = \begin{pmatrix} z \\ x \end{pmatrix}, p_u = \begin{pmatrix} z_u \\ x_u \end{pmatrix}.$$

Here n_r and n_z denote the number of real and integer variables and $x \in \mathbb{R}^{n_r}, z \in \mathbb{Z}^{n_z}$ and $c(p) : \mathbb{R}^{n_r+n_z} \to \mathbb{R}^m$. When integer variables are present, a popular first choice class of algorithms to explore are Evolutionary Algorithms (EAs) or a subset of EAs called Genetic Algorithms (GAs). Further, such algorithms perform a global search of the feasible region and hence seek out robust minimums. A well-known drawback is that the number of function evaluations can be exorbitant.

The objective of this work is to improve the capabilities of the GSS algorithm to handle integer and categorical variables within a hybrid optimization framework. Recently, frameworks such as those used in [20, 48, 21, 45], were developed to fascillitate the rapid natural construction of loosely coupled asynchronous parallel hybrid optimization algorithms, which exploit load-balance to create robust algorithms with improved run-time. The structure ensures that existing theoretical convergence properties of individual algorithms are maintained, while permitting individuals to monitor and leverage real-time performance of other concurrently running individuals. Given a sufficient number of processors, the hybrid algorithm typically is no slower, than the fastest algorithm contained within the hybrid. Thus, if the hybrid contains optimization algorithms A, B, and C, if A is the fastest for a given problem, then Hybrid(A,B,C) is usually as fast as algorithm A. A tremendous benefit to the analyst, is that there is no need for the user (often having limited optimization background) to predict beforehand which of the existing solvers will be the most suitable for a given optimization problem.

We construct a hybrid algorithm which naturally extends GSS to the mixed-integer environment by using existing state of the art genetic algorithm software. We refer to the software used to implement the hybrid algorithm framework as Evolutionary Algorithms Guiding Local Search (EAGLS). Hybrid combinations of a GA with a local search method are not new. See for example, [4, 25] and the references therein. However, here we use the GA as an outer optimization tool and make use of the binary capabilities to handle the integer variables while GSS performs a local search on the real variables only. In this context, the GSS can do what it does best, local derivative-free refinement, while the GA can perform a global search, and moreover the integer variables are handled naturally.

The primary purpose of this paper is to provide a natural extension of popular direct search algorithms, currently restricted to real variable problems, to optimization problems that have (non-relaxable) binary variables in a manner where very little is required by the end user. We proceed by describing the GA and GSS algorithms and the software used to implement the hybrid algorithm (the NSGA-II [11, 52, 6, 9] and APPSPACK [28, 30]). Fortunately NSGA-II and APPSPACK are two well-known software packages that can often be used "out of the box". Thus these two software packages were a natural choice for parallel hybrid optimization, where each algorithm is free to do what it does best: (1) the GA's handling of integer and real variables for global search, and (2) the GSS's handling of real variables in parallel for local search. In sections 4 and 5 we describe the hybridization of the GA and GSS. In section 6 we present results for a compression spring model [3], a simulation-based hydrology application [35, 27, 18, 26], and a standard mixed-integer test problem and point the way towards future work.

2 Genetic Algorithms

Genetic algorithms are part of a larger class of evolutionary algorithms and are classified as population based, global search heuristic methods [16]. Genetic algorithms are based on biological processes such as survival of the fittest, natural selection, inheritance, mutation, and reproduction. Design points are coded as "individuals" or "chromosomes", typically as binary strings, in a population. Through the above biological processes, the population evolves through a user specified number of generations towards a smaller fitness value. We can define the following simple GA in Algorithm 1.

Algorithm 1 Genetic Algorithm Framework

```
Require: Population size $n_p$, Number of Generations $n_g$
1: Generate initial population, determine fitness, and rank : $P_1 = p_1, \dots, p_{n_p}$
2: for $k = 1, \dots, n_g$ do
3:     $P_{k+1} = \text{select}(P_k)$
4:     $P_{k+1} = \text{crossover}(P_{k+1})$
5:     $P_{k+1} = \text{mutate}(P_{k+1})$
6:     Determine fitness for $P_{k+1}$
7: end for
```

We should note that GAs can directly handle bound constraints but linear and nonlinear constraints are often included in Eq. (1.1) by using a barrier or penalty approach. We describe our implementation in section 6. Thus the fitness of an individual may include the objective function along with a measure of constraint violation. 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 prevent the algorithm from terminating prematurely to a suboptimal point and is used as a means to explore the design space. Termination of the algorithm is based on a prescribed number of generations or when the highest ranked individual's fitness has reached a plateau.

Genetic algorithms are often criticized for their computational complexity and dependence on optimization parameter settings, which are not known a priori. However, if the user is willing to exhaust a large number of function evaluations, the GA can help gain insight into the design space and locate initial points for fast, local single search methods.

In this work, we use the Non-dominated Sorting Genetic Algorithm NSGA-II, which is described in [11, 52, 6, 9]. 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 including the two hydrology applications used in this work. Here, we consider a single-objective use of the NSGA-II, which incorporates both real- and binary-coded variables, and uses binary tournament selection [7]. We use the standard dual binary system representation for the integer valued variables. For the real-coded variables, the simulated binary crossover (SBX) operator [7, 8] with polynomial mutation is used while single-point crossover with bitwise mutation are used for binary-coded variables. 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 [46, 36].

3 Generating Set Search and Pattern Search

Generating set search (GSS) denotes a class of algorithms for bound and linearly constrained optimization problems that obtain conforming search directions from generators of local tangent cones [32, 31]. In the case that only bound constraints are present, GSS is identical to a pattern search optimization algorithm. Pattern searches use a predetermined pattern of points to sample a given function domain. It has been shown that if certain requirements on the form of the points in this pattern are followed and if the objective function is suitably smooth, convergence to a stationary point is guaranteed [13, 33, 50].

The majority of the computational cost of pattern search methods is the function evaluations, so parallel pattern search (PPS) techniques have been developed to reduce the overall computation time. Specifically, PPS exploits the fact that once the points in the search pattern have been defined, the function values at these points can be computed simultaneously [12, 49]. In this work, we specifically consider Asynchronous Parallel Pattern Search (APPS) [28, 30]. The APPS algorithm is a modification of PPS that eliminates the synchronization requirements. It retains the positive features of PPS, but reduces processor latency and requires less total time than PPS to return results [28]. Implementations of APPS have minimal requirements on the number of processors and do not assume that the amount of time required for an objective function evaluation is constant or that the processors are homogeneous.

Omitting the implementation details, the basic APPS algorithm can be simply outlined as follows:

1. Generate a set of trial points to be evaluated.

2. Send the set of trial points to the *conveyor* for evaluation, and collect a nonempty set of evaluated points from the conveyor. (The conveyor is a mechanism for shuttling trial points through the process of being evaluated.)

3. Process the set of evaluated points and see if it contains a new *best point*. If there is such a point, then the iteration is successful; otherwise, it is unsuccessful.

4. If the iteration is successful, replace the current best point with the new best point. Optionally, regenerate the set of search directions and delete any pending trial points in the conveyor.

5. If the iteration is unsuccessful, reduce certain step lengths as appropriate. In addition, check for convergence based on the step lengths.

A detailed procedural version of APPS is given in [19], and a complete mathematical description and analysis is available in [30].

The APPS algorithm described here has been implemented in an open source software package called APPSPACK. It is written in C++ and uses MPI [23, 24] for parallelism. The details of the implementation are described in [19]. APPSPACK has been successfully applied to problems in microfluidics, biology, groundwater, thermal design, and forging. See [19] and references therein.

APPSPACK performs function evaluations through system calls to an external executable which can written in any computer language. This simplifies its execution and makes it amenable to customization. Of particular interest to us is the management of the function evaluation process. The procedure is quite general and merely one way of handling the process of parallelizing multiple independent function evaluations while efficiently balancing computational load. This management system makes APPSPACK particularly amenable to hybridization in that it can readily accommodate externally generated points.

$\boxed{4}$ Trial-point Selection Criteria for Parallel Local Search

For the simultaneous local search, it is important to avoid starting points that correspond to the same local minimizer. Rather, as shown in Figure 4.1, it is preferable to start from points that are both promising with respect to their current objective value, but also diversely located so as to increase the likelihood of finding different local minimums.

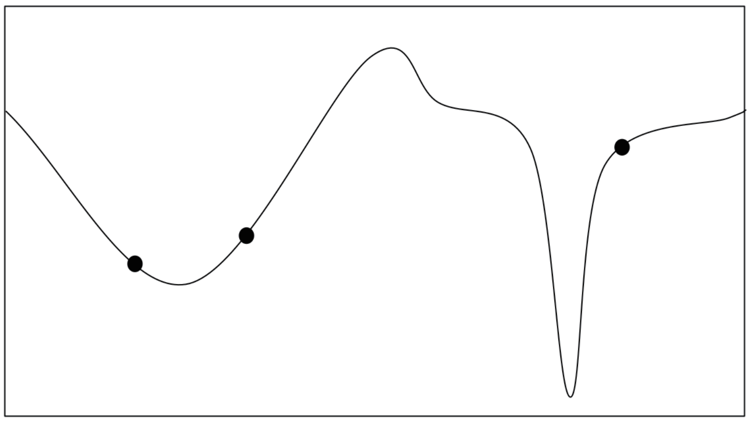

Figure 4.1: For a nonconvex function, the point having the best fitness score is not necessarily the best starting point for local optimization. In this example, (supposing there are only resources available to maintain two parallel local searches) starting a local search at p_1 and p_3 is preferable to beginning from p_1 and p_2.

In this paper we apply a simple strategy that seeks to leverage the presence of *niching* techniques within genetic algorithms [5, 17, 44, 34].The need for niching arises in genetic algorithms applied to multimodal optimization since the tendency is for the population to converge to a single optimum. Further, little is gained by merging points which belong to different basins of the decision variable space. Thus niching is a commonly used strategy to naturally divide the current population into sub-populations (or demes) corresponding to different minimas [2]. The essential idea is to rank the population P_k not only with respect to the corresponding fitness score, but also with respect to some metric measuring similarity (or dissimilarity) between points, such as the *crowding distance.* The genetic algorithm is then modified to encourage sub-populations to remain in their respective deme.

Given the resource for k parallel searches, ideally we would spread these equally over the most promising demes. If we assume that the underlying GA has been constructed to support some variant of niching then we can implicitly encourage points to come from distinct demes by using a simple scheme that penalizes points that are geometrically close to nearby points with better fitness scored. We first define the distance, d, between two points using the scaled-Euclidean distance

$$d(v,w) = \sqrt{\sum_{i=1}^{n_r+n_z} s_i \left(\frac{v_i - w_i}{(p_\ell)_i - (p_u)_i}\right)^2}; \tag{4.1}$$

and then rank points by recursively selecting the best point from the population penalized by lying with close proximity to previously selected promising points.

$$\begin{aligned} p'_1 &= \arg\min\{f(p) + \mathcal{P}(p,\rho,\alpha) \mid p \in P_k\} \\ p'_2 &= \arg\min\{f(p) + \mathcal{P}(p,\rho,\alpha) + \gamma \frac{1}{d(p,p'_1)} \mid p \in P_k\} \\ &\vdots \\ p'_{n_p} &= \arg\min\{f(p) + \mathcal{P}(p,\rho,\alpha) + \gamma \sum_{j=1}^{n_p-1} \frac{1}{d(p,p'_j)} \mid p \in P_k\} \end{aligned}$$

The penalization parameter γ can then be used to encourage starting point selection to come from diverse members of the population pool. There are of course more sophisticated schemes that may be used (i.e. explicitly select the best point from deme for local optimization); however, we have found that this simple sorting scheme to be sufficient for our needs. Note further that if s_i equals one for integer variables and zero for continuous variables, then this scheme will ensure that each instance in the parallel local search pool is refining a separate integer plane as shown in Figure 5.1.

5 EAGLS Algorithm Description

In Algorithm 2 we present a synchronous version of the EAGLS algorithm to simplify the description of the algorithm which in practice was implemented asynchronously using concepts similar to those exploited in [20, 48, 21]. In synchronous mode the algorithm essentially nests a GA iteration with a parallel local search refinement phase (similar strategies have been proposed in [47, 51, 10]).

In Step 3-Step 7 the classic steps of a genetic algorithm are executed. Step 8-Step 9 are dedicated to the local search phase of the algorithm. Note that in practice these steps are happening simultaneously to prevent load imbalance; to ensure that GA iterations are spliced by local search iterations, a priority queuing system is used to place new population of trial points lower in the evaluation queue until the current local search evaluation budget has been expended. Note that we only refined evaluated parent points in the populations. Unevaluated child trial-points cannot be selected for local refinement. This prevents a child point from drifting before it can be evaluated.

Algorithm 2 Genetic Algorithm Guiding Local Search

```
Require: Population size: $n_p$
Require: Maximum number of generations: $n_g$
Require: Budget for local search: $n_b$
Require: Number of parallel local searches desired: $n_s$
 1: Generate (evaluate in parallel) and rank initial population: $P_1 = p_1, \dots, p_{n_p}$
 2: for $k = 1, \dots, n_g$ do
 3:     $P_{k+1} = \text{select}(P_k)$
 4:     $P_{k+1} = \text{mutate}(P_{k+1})$
 5:     Evaluate in parallel new points in $P_{k+1}$
 6:     $P_{k+1} = \text{merge}(P_k, P_{k+1})$
 7:     $P'_{k+1} = \text{rank}(P_{k+1})$
 8:     Choose first $n_s$ of $P'_{k+1}$ for local search
 9:     Create $n_s$ instances of APPS for $1 \le i \le n_s$ subproblems:
            $$\begin{array}{rl} \underset{x \in \mathbb{R}^{n_r}}{\text{minimize}} & f(x, \text{int}(p'_i)) \\ \text{subject to} & x_\ell \le x \le x_u \end{array} \qquad (5.1)$$
10:     while number of evaluations $< n_b$ do
11:         Run APPS instance in parallel with parallel evaluations
12:     end while
13: end for
14:
```

The variant of EAGLS we describe in Algorithm 2 has three point of synchronization which can be sources of load imbalance: Step 1 and Step 5 where new point generated by the GA must be evaluated, and Step 9 where the n_s solvers must either find local solutions or expend their collective budgets prior to exiting. These points of synchronization can readily be removed if a shared evaluation queue is used as described in [21]; at which point the local search can continue to run while the GA is waiting for Step 5 to complete. This is helpful when executing function evaluation in parallel that are computational expensive and have widely varied evaluation times.

The combinatorial nature of integer variables make these problems very difficult. If the integer variables are relaxable (i.e. the objective function is defined for rational variables) more sophisticated schemes such as branch and bound may be preferred options. However, for simulation based optimization problems, often the integer variables are categorical having no natural ordering and hence cannot be relaxed. That is, there is no well-defined mathematical definition for what is meant by "nearby".

One possibility is to manually create definitions for neighborhoods such as [1]; these neighborhoods can then be used to allow the direct search algorithm to sample "nearby" integer planes. In this approach, for a point to be declared "optimal", it must satisfy the algorithms criteria for local optimality with respect to the continuous variables, and no worse than the best point in the corresponding neighborhood. The user must therefore balance efficiency (if the selected neighborhood definitions are too large the algorithm may need to evaluate many points) with effectiveness (if the neighborhood definitions are too restrictive, the global solution may not be obtainable, regardless of the number of iterations).

Another possibility for a such problem is to attempt to learn a smart mapping that somehow dynamically learns the ordering of categorical variables using surrogate models such as [26, 43]; this mapping can then be used to recover the notion of neighborhood for discrete variables, as the surrogate smoothly extends the integer variables to the real variables. One drawback to using surrogates is the time to create a model can be prohibitive as the number of evaluations increase.

Our paper explores a third possibility, where the genetic algorithm is used to move from one integer plane to another (see Figure 5.1). In a sense, the genetic algorithm defines a probabilistic-based mapping from one integer plane to the next, while the local search corrects for errors in the mapping. Of course, if the user can provide information or a metric for changes in the integer variables, this could be used to set the parameters and operator options of the GA appropriate to this.

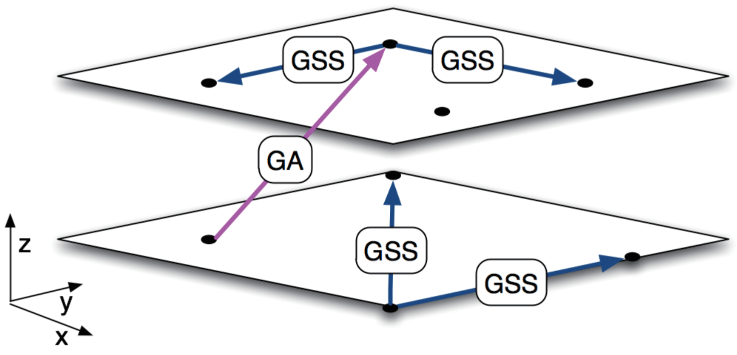

Figure 5.1: While the local searches are constrained to a given integer plane with the integer component fixed, the genetic algorithm is free to move points both with respect to the real variables x, y, and the integer variables z.

$\boxed{6}$ Numerical Results

We demonstrate the performance of EAGLS on three mixed-integer problems. One is from a spring mass model, one is a simulation based problem from hydrology, and one is a standard MINLP test problem taken from [29]. All problems include nonlinear inequality constraints. A popular strategy for handling constraints is to use penalty function where the constraint violation is incorporated with the objective function to form a corresponding merit function. In this paper we consider the ℓ_1 and the ℓ_1-smoothed penalty function

$$s\mathcal{P}(p,\rho) = \rho\sum_{i=1}^{m}\max(0,c_i(p)) \tag{6.1}$$

$$\mathcal{P}(p,\rho,\alpha) = \alpha\sum_{i=1}^{m}\ln(1+e^{\rho c_i(p)/\alpha}). \tag{6.2}$$

It can be shown that

$$\mathcal{P}(p,\rho) \le \mathcal{P}(p,\rho,\alpha) \le \mathcal{P}(p,\rho) + 3m\alpha.$$

whose effectiveness was explored in detail in [22]. Thus, independent of the choice of ρ (which, in practice, is often quite large),

$$|\mathcal{P}(p,\rho) - \mathcal{P}(p,\rho,\alpha)| \le 3m\alpha,$$

and hence the approximation level is uniformly controlled by the size of α. The ℓ_1 penalty function is attractive as one can show under certain standard assumption that a minimizer of the constrained problem coincides with an unconstrained minimizer of the ℓ_1 penalty function when ρ is sufficiently large. Thus the constrained problem reduces to an unconstrained problem, if ρ is selected appropriately.

Unfortunately, for primal approaches, exact penalty function are necessarily nonsmooth. As stated early, direct search algorithms such as GSS, tie convergence (albeit loosely) to an implicit assumption that the objective function, modulo noise, is smooth. Thus, GSS easily gets stuck in practice at a nonsmooth point. To avoid this difficulty, a smoothed variant of the exact penalty function may be used, where ρ is sufficiently large with respect to the current step size, to prevent stagnation of the algorithm. Thus in our approach the GA will optimize Eq. (6.1) independently from GSS, where the GA (being less susceptible to real nonsmoothness) will use the exact penalty function $\mathcal{P}(x,\rho)$ directly , while the GSS will leverage the smoothed variant $\mathcal{P}(x,\rho,\alpha)$.

$\boxed{6.1}$ Compression Spring Application

This problem is based on a problem presented in [42]. This objective of this problem is to design a coil compression spring with minimum volume. See Figure 6.1.

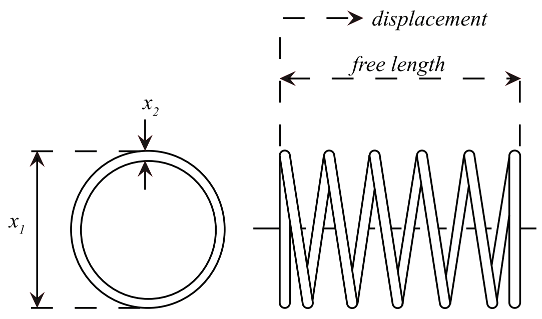

Figure 6.1: Compression spring problem

The decision variables for this problem are $p = (z_1, x_1, x_2)^T$ where z_1 is the number of coils, x_1 is the coil diameter, and x_2 is the wire diameter (measured in inches). The bound constraints are given by

$$p \in \Omega = \{p | z \in \{1, 2, \ldots, 70\}, x_1 \in [0.6, 3], x_2 \in [0.207, 0.5]\}.$$

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

$$f(p) = \pi^2 \frac{x_1 x_2^2 (z_1 + 2)}{4} \tag{6.3}$$

subject to the following constraints,

$$\begin{array}{l} c_1(p) = \frac{SC_f F_{max} x_1}{\pi x_2^3} - S \leq 0 \\ c_2(p) = l_f - l_{max} \leq 0 \\ c_3(p) = \sigma_p - \sigma_{pm} \leq 0 \\ c_4(p) = \sigma_p - \frac{F_p}{K} \leq 0 \\ c_5(p) = \sigma_w - \frac{F_{max}-F}{K} \leq 0. \end{array} \tag{6.4}$$

The parameters are defined by

$$\begin{array}{l} C_f = 1 + 0.75 \frac{x_2}{x_1 - x_2} + 0.615 * \frac{x_2}{x_1} \\ F_{max} = 1000 \\ S = 189000 \\ l_f = \frac{F_{max}}{K} + 1.05 * (z_1 + 2) * x_2 \\ l_{max} = 14; \\ \sigma_p = \frac{F_p}{K} \\ \sigma_{pm} = 6 \\ F_p = 300 \\ K = 11.5 \times 10^6 \frac{x_2^4}{8 z_1 x_1^3} \\ \sigma_w = 1.25 \end{array} \tag{6.5}$$

Due to the stochastic nature of the genetic algorithm, we ran 500 optimization trials. The published solution to this problem in [3] has function value of 2.6254 although for that work, x_2 is also treated as a discrete variable, whereas x_2 is treated as a continous variable here. The best point found for this work was $p = (6, 1.49881021096990, 0.297845552041890)$ which gives a function value of 2.6246. In the discussion that follows, let $j = 1, \ldots, 500$ denote the counter for the optimization trials and let i denote each individual optimization trial's function evaluation counter. We consider a run successful if the function value is within 1% of the best function value found, denoted f_b, in all 500 optimization experiments. This approach is comparable to that proposed in [41] where instead each heuristic optimization trial would be considered a different benchmarking problem.

Moreover, since we allowed EAGLS a large function evaluation budget to allow for an exhaustive search of the design space, we normalize our function evaluation counter by identifying the most iterations it took for any optimization trial to be successful. Specifically, we consider $i/\max\{l_j\}$, where $l_j = min\{i|f_{i,j} \leq f_b\}$ indicates the function evaluation when success was detected. For this problem, the slowest convergence of any trial took 1364 function evaluations.

We demonstrate two performance measures, shown in Figure 6.2. The left vertical axis shows the lowest and highest function values found as the optimization progresses. The right vertical axis shows the success rate, that is the fraction of of successful runs defined by

$$\frac{\{j : f_{i,j} \leq f_b\}}{max\{l_j\}}. \tag{6.6}$$

The steep jump around 0.4 means that within 40 % of the most function evaluations needed for success, just below 80% of the optimization runs had already successfully found the solution. This further implies that although the GA is stochastic in nature, there is a high probability of success in terms of locating the optimal point within 1000 function evaluations for this problem.

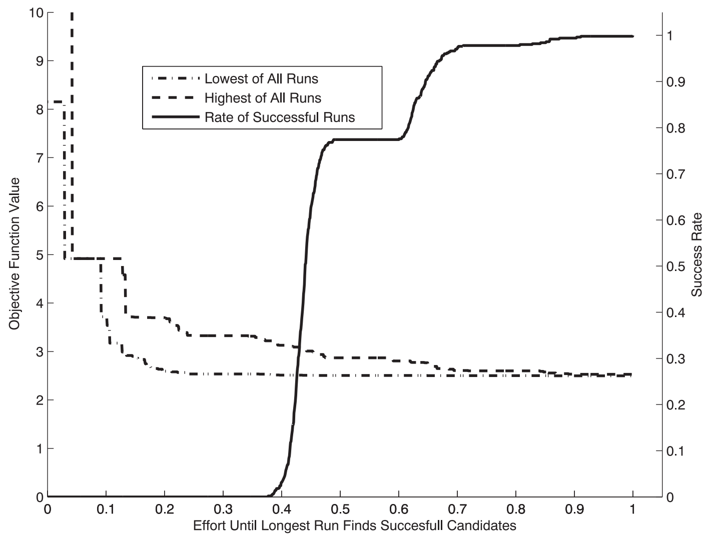

Figure 6.2: Compression spring problem results

$\boxed{6.2}$ Hydrology Application

We consider a water supply problem, which is described in [37, 38] and have been studied in detail as a benchmarking problem for optimization methods [27, 14, 15, 26]. The objective is to minimize the cost needed to install and operate a well-field that supplies a specified quantity of water. The decision variables are the pumping rates $\{Q_k\}_{k=1}^n$, the well locations $\{(x_k, y_k)\}_{k=1}^n$, and the number of wells $n \leq N_w$. Here n is the number of wells in the final design and $N_w = 6$ is the maximum number of wells allowed. We determine the number of wells by using a binary indicator to indicate if a well is on or off in the final design. We proceed by briefly describing the objective function and constraints.

We consider a capital cost f^c to install wells and a cost f^o to operate these wells, and we seek to minimize the total cost $f^T = f^c + f^o$. A negative pumping rate means that a well is extracting and a positive pumping rate means that a well is injecting. The objective function, as proposed in [37, 38] is given by

$$f^T = \underbrace{\sum_{k=1}^{n} s_k c_0 d_k^{b0} + \sum_{k, Q_k < 0.0} s_k c_1 |Q_k^m|^{b_1} (z_{gs} - h^{min})^{b_2}}_{f^c} \tag{6.7}$$

$$+ \underbrace{\int_0^{t_f} \left(\sum_{k, Q_k < 0.0} s_k c_2 Q_k (h_k - z_{gs}) + \sum_{k, Q_k > 0.0} s_k c_3 Q_k \right) dt}_{f^o},$$

where c_j and b_j are cost coefficients and exponents, $d_k = z_{gs}$ is the depth of well k, Q_k^m is the design pumping rate for which we use $Q_k^m = 1.5 Q_k$ m^3/s, h^{min} is the minimum allowable hydraulic head, and h_k is the hydraulic head in well k. In Eq. (6.7), $s_k = 0, 1$

indicates whether or not a well is on or off. Obtaining the hydraulic head values requires a call to a groundwater flow simulator for a solution to a partial differential equation that models saturated groundwater flow. For this work, we used the U.S. Geological Survey code MODFLOW-96 [39, 40]. MODFLOW is a widely used and well supported block-centered finite difference code that simulates saturated groundwater flow. The cost coefficients are given in [38].

We constrain the pumping rates and hydraulic head for the objective function given in Eq. (6.7). The constraints are given by

$$-0.0064 \text{ m}^3/\text{s} \leq Q_k \leq 0.0064 \text{ m}^3/\text{s}, k = 1, ..., n, \text{ and} \tag{6.8}$$

$$10 \text{ m} \leq h_k \leq 30 \text{ m}, k = 1, ..., n. \tag{6.9}$$

Constraints (6.8) and (6.9) are enforced at each well. Constraint (6.8) reflects physical limits on the pumps and well design. We also constrain the net pumping rate. We specify the amount of water to supply with

$$\sum_{k=1}^{n} Q_k \leq -0.032 \text{ m}^3/\text{s}. \tag{6.10}$$

Since this constraint depends only on the pumping rates, it is checked first and if the inequality is not satisfied, the simulator in not executed.

We allowed 50 optimization trials and as in the previous discussion on the compression spring model, we let j denote the optimization trial and i denote the function evaluations counter. For all of the hydrology applications, the installation of a well is roughly \$20,000 while the operating cost is only about \$1,000 per year, thus the integer variable that defines the installation of a well leads to a large decrease in cost.

The water supply problem has been shown to be particularly challenging [15, 14] because to satisfy the supply constraint given by Eq. (6.10), the solution requires 5 wells pumping at the maximum allowable extraction rate given in Eq. (6.8). Thus to test the integer and local search capabilities of EAGLS, we allow for at most 6 wells which means that the only possible feasible points would have 5 or 6 active wells. It was shown in [15] there are many local minima with 6 wells and for the optimization methods tested there (which included APPSPACK and the NSGA-II as stand alone optimization approaches), an initial 6 well design with at least 5 wells with rates set to -0.0064 m^3/s was required for convergence. Published solutions for this problem have a final objective function value of roughly \$124,000 and EAGLS was able to locate a comparable point without any initial iterate, which is a significant improvement over the performance of previous attempts by both APPSPACK and the NSGA-II. Figure 6.3 shows the performance of EAGLS for this application. Here we again define a successful optimization run as resulting in a function value within 1% of the best value found over all. It took the slowest optimization trial only 65 function evaluations to detect success. We allowed EAGLS a budget of 3000 function calls and of this only around 1000 satisfied the linear constraint given by Eq. (6.10), which means that the simulator was not executed. Although there is not a steep jump in terms of the rate of success, the highest function value of all optimization runs drops down to roughly \$127,000 after about 20% of the function evaluations are performed (roughly 13), meaning that a five well design with the appropriate pumping rates was found and from that point forward the optimal locations are being sought to improve the operational cost in Eq (6.7).

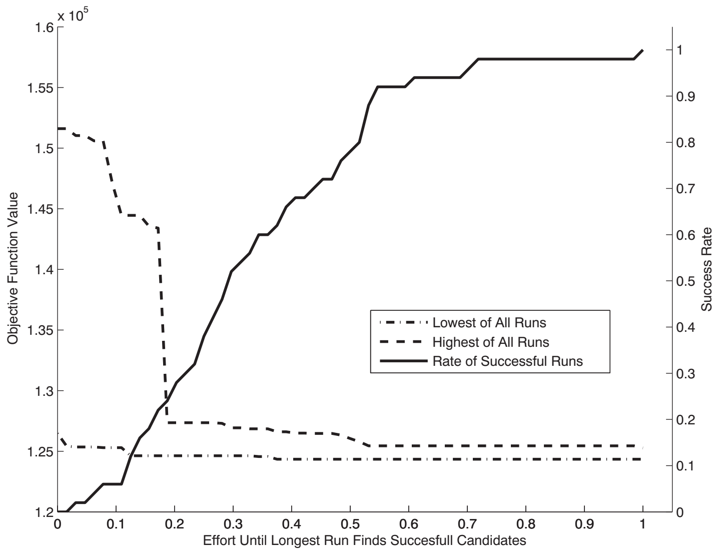

Figure 6.3: Water supply problem results

$\boxed{6.3}$ Standard Test Problem

This test problem is taken from [29] and t he 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

$$f(p) = 2x_1 + 3x_2 + 1.5z_1 + 2z_2 - 0.5z_3 \tag{6.11}$$

subject to the following constraints,

$$\begin{aligned} c_1(p) &= x_1^2 + z_1 - 1.25 = 0 \\ c_2(p) &= x_2^{1.5} + 1.5z_2 - 3.00 = 0 \\ c_3(p) &= x_1 + z_1 - 1.60 \leq 0 \\ c_4(p) &= 1.333x_2 + z_2 - 3.00 \leq 0 \\ c_5(p) &= -z_1 - z_2 + z_3 \leq 0. \end{aligned} \tag{6.12}$$

We used the same performance measures as above and 500 optimization trials. The convergence results are shown in Figure 6.4. It took the slowest optimization run 772 function evaluations to find the solution. Note that after about 0.7 on the horizontal axis, indicating about 540 function calls, nearly all optimization runs had located the solution.

$\boxed{7}$ Conclusions

We have provided a parallel framework for using an evolutionary algorithm to extend the capabilities of a local generating set-search method to handle integer and categorical variables, while help to globalize the search for good minima. The implementation presented

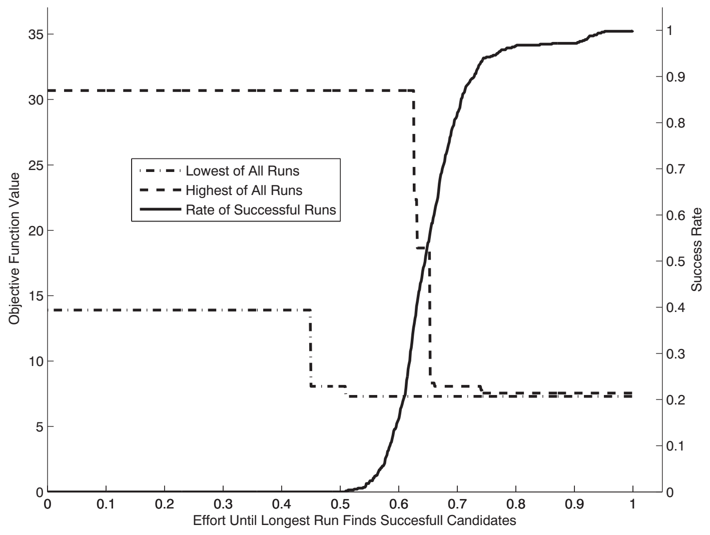

Figure 6.4: Standard MINLP test problem results

here combined two software packages, the NSGA-II and APPSPACK. For each of the three test problems, multiple optimization trials were run to understand the success of the algorithm since the GA is not deterministic. The numerical results on three mixed-integer problems are promising and solutions obtained are comparable to those found in the literature for all problems. EAGLS is currently under further development and future directions include developing appropriate stopping criteria, more advanced constraint handling, guidance in selecting algorithmic parameters, and further testing on problems with categorical variables.

References

[1] M.A. Abramson, C. Audet, J.W. Chrissis and J.G. Walston, Mesh adaptive direct search algorithms for mixed variable optimization, *Optimization Letters* 3 (2007) 35–47.

[2] E. Cantu-Paz, *A Survey of Parallel Genetic Algorithms*, PhD thesis, University of Illonois, May 1997.

[3] M. Clerc, A method to improve standard pso, Tech. Report MC2009-03-13, 2009.

[4] J. Contact and G.R. Raidl, *Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification*, Lecture Notes in Computer Science, Springer, Berlin, 2005.

[5] K.A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, master's thesis, Ann Arbor, MI, USA, 1975.

[6] K. Deb, An efficient constraint handling method for genetic algorithms, *Computer Methods in Applied Mechanics and Engineering* 186 (2000) 311–338.

[7] K. Deb and R.B. Agrawal, Simulated binary crossover for continuous search space, *Complex Systems* 9 (1995) 115–148.

[8] K. Deb and H.G. Beyer, Self-adaptive genetic algorithms with simulated binary crossover, *Evolutionary Computation Journal* 9 (2001), pp. 197–221.

[9] K. Deb and T. Goel, Controlled elitist non-dominated sorting genetic algorithms for better convergence, in *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization EMO 2001*, E. Zitler, K. Deb, L. Thiele, C. Coello-Coello and D. Corne (eds.), Lecture Notes on Computer Science, 2001, pp. 67–81.

[10] K. Deb, S. Lele and R. Datta, A hybrid evolutionary multi-objective and sqp based procedure for constrained optimization, *Lecture Notes in Computer Science* 4683 (2007) 36–45.

[11] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, A fast and elitist multi-objective genetic algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation* 6 (2002) 182–197.

[12] J.E. Dennis, Jr. and V. Torczon, Direct search methods on parallel machines, *SIAM J. Optim.* 1 (1991) 448–474.

[13] E.D. Dolan, R.M. Lewis and V. Torczon, On the local convergence properties of parallel pattern search, Tech. Report 2000-36, NASA Langley Research Center, 2000.

[14] K.R. Fowler, C.T. Kelley, C.T. Miller, C.E. Kees, R.W. Darwin, J.P. Reese, M.W. Farthing and M.S.C. Reed, Solution of a well-field design problem with implicit filtering, *Optimization and Engineering* 5 (2004) 207–234.

[15] K.R. Fowler, J.P. Reese, C.E. Kees, J.E. Dennis, Jr., C.T. Kelley, C.T. Miller, C. Audet, A.J. Booker, G. Couture, R.W. Darwin, M.W. Farthing, D.E. Finkel, J.M. Gablonsky, G. Gray and T. G. Kolda, A comparison of derivative-free optimization methods for water supply and hydraulic capture community problems, Accepted to Advances in Water Resources, (2008).

[16] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley Pub. Company, 1989.

[17] D.E. Goldberg and J. Richardson, Genetic algorithms with sharing for multimodal function optimization, in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, Hillsdale*, NJ, USA, 1987, L. Erlbaum Associates Inc., pp. 41–49.

[18] G.A. Gray and K.R. Fowler, Approaching the groundwater remediation problem using multifidelity optimization, in *Proc. of the CMWR XVI - Computational Methods in Water Resources*, 19-22 June 2006.

[19] G.A. Gray and T.G. Kolda, Algorithm 856: APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization, *ACM TOMS* 32 (2006) 485–507.

[20] G.A. Gray, M. Taddy, M. Martinez-Canales and H.K.H. Lee., Enhancing parallel pattern search optimization with a gaussian process oracle, in In Proceedings of the 14th Nuclear Explosive Codes Development Conference (NECDC), 2007.

[21] J.D. Griffin and T.G. Kolda, Asynchronous parallel hybrid optimization combining direct and gss, *Optimization Methods and Software* (2009).

[22] J.D. Griffin and T.G. Kolda, Nonlinearly-constrained optimization using heuristic penalty methods and asynchronous parallel generating set search, Applied Mathematics Research eXpress, (2010). in press.

[23] W. Gropp, E. Lusk, N. Doss and A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, *Parallel Comput.* 22 (1996) 789–828.

[24] W.D. Gropp and E. Lusk, User's guide for `mpich`, a portable implementation of MPI, Tech. Report ANL-96/6, Mathematics and Computer Science Division, Argonne National Lab, 1996.

[25] W. Hart, *Adaptive Global Optimization with Local Search*, PhD thesis, University of California, San Diego, CA, 1994.

[26] T. Hemker, K.R. Fowler, M.W. Farthing and O. von Stryk, A mixed-integer simulation-based optimization approach with surrogate functions in water resources management, Accepted to Optimization and Engineering, (2007).

[27] T. Hemker, K.R. Fowler and O. von Stryk, Derivative-free optimization methods for handling fixed costs in optimal groundwater remediation design, in *Proc. of the CMWR XVI - Computational Methods in Water Resources*, 19-22 June 2006.

[28] P.D. Hough, T.G. Kolda and V. Torczon, Asynchronous parallel pattern search for nonlinear optimization, *SIAM J. Sci. Comput.* 23 (2001) 134–156.

[29] G. Kocis and I. Grossmann, Global optimization of nonconvex mixed-integer nonlinear programming (minlp) problems in process synthesis, *Ind. Eng. Chem. Res.* 27 (1988) 1407–1421.

[30] T.G. Kolda, Revisiting asynchronous parallel pattern search for nonlinear optimization, *SIAM J. Opt.* 16 (2005).

[31] T.G. Kolda, R.M. Lewis and V. Torczon, Stationarity results for generating set search for linearly constrained optimization, *SIAM J. Optim.* 17 (2006) 943–968.

[32] R.M. Lewis, A. Shepherd and V. Torczon, Implementing generating set search methods for linearly constrained minimization, Tech. Report WM-CS-2005-01, Department of Computer Science, College of William & Mary, Williamsburg, VA, July 2005. Revised July 2006.

[33] R.M. Lewis and V. Torczon, Rank ordering and positive basis in pattern search algorithms, Tech. Report 96-71, NASA Langley Research Center, Inst. Comput. Appl. Sci. Engrg., Hampton, VA, 1996.

[34] J.-P. Li, M.E. Balazs, G.T. Parks and P.J. Clarkson, A species conserving genetic algorithm for multimodal function optimization, *Evol. Comput.* 10 (2002) 207–234.

[35] S.L. Mattot, A.J. Rabideau and J.R. Craig, Optimization of pump and treat systems using analytic element flow models, *Advances in Water Resources* 29 (2006) 760–775.

[36] A. Mayer, C. Kelley and C. Miller, Optimal design for problems involving flow and transport phenonmena in saturated subsurface systems, *Advances in Water Resources* 12 (2002) 1233–1256.

[37] A.S. Mayer, C.T. Kelley and C.T. Miller, Optimal design for problems involving flow and transport phenomena in saturated subsurface systems, *Advances in Water Resources* 12 (2002) 1233–1256.

[38] A.S. Mayer, C.T. Kelley and C.T. Miller, Electronic supplement to "Optimal design for problems involving flow and transport phenomena in saturated subsurface systems", 2003. "http://www.elsevier.com/gej-ng/10/8/34/58/59/41/63/show/index.htt", 17 pages.

[39] M.G. McDonald and A.W. Harbaugh, A modular three dimensional finite difference groundwater flow model, U.S. Geological Survey Techniques of Water Resources Investigations, (1988).

[40] M.G. McDonald and A.W. Harbaugh, Programmer's documentation for MODFLOW-96, an update to the U.S. geological survey modular finite difference groundwater flow model, U.S. Geological Survey Open-File Report 96-486, (1996).

[41] J.J. More and S.M. Wild, Benchmarking derivative-free optimization algorithms, *SIAM Journal of Optimization* 20 (2009) 172–191.

[42] G. Onwubolu and B. Babu, *New Optimization Techniques in Engineering*, Springer, Berlin, Germany, 2004.

[43] M.D. Parno, K.R. Fowler and T. Hemker, Framework for particle swarm optimization with surrogate functions, Tech. Report TUD-CS-2009-0139, Technische Universität Darmstadt, Department of Computer Science, 2009.

[44] A. Petrowski, A clearing procedure as a niching method for genetic algorithms, in *Proceedings of Third IEEE International Conference on Evolutionary Computation(ICEC'96)*, Piscataway, NJ, 1996, IEEE Press, pp. 798–803.

[45] T.D. Plantenga, Hopspack 2.0 user manual, Tech. Report SAND2009-6265, Informatics and Decisions Science Department, Sandia National Laboratories, 2009.

[46] P. Reed, B. Minsker and D. Goldberg, Designing a competent simple genetic algorithm for search and optimization, *Water Resources Research* 36 (2000) 3757–3761.

[47] M. Sayeed, An efficient parallel optimization framework for inverse problems, PhD thesis, North Carolina State University, 2004. Director-Mahinthakumar, G.

[48] M.A. Taddy, H.K.H. Lee, G.A. Gray and J. Griffin, Bayesian guidance for robust pattern search optimization, *Technometrics* 51 (2009) 389–401.

[49] V. Torczon, PDS: Direct search methods for unconstrained optimization on either sequential or parallel machines, Tech. Report TR92-09, Rice Univ., Dept. Comput. Appl. Math., Houston, TX, 1992.

[50] V. Torczon, On the convergence of pattern search algorithms, *SIAM J. Optim.* 7 (1997) 1–25.

[51] A. Ẑilinskas and J. Ẑilinskas, Parallel hybrid algorithm for global optimization of problems occurring in mds-based visualization, *Comput. Math. Appl.* 52 (2006) 211–224.

[52] E. Zitzler, K. Deb and L. Thiele, Comparison of multiobjective evolutionary algorithms: Empirical results, *Evolutionary Computation Journal* 8 (2000) 173–195.

Manuscript received 14 April 2010
revised 30 August 2010
accepted for publication 10 September 2010

J.D. Griffin
SAS Institute, Inc Raleigh, NC
E-mail address: `joshua.griffin@sas.edu`

K.R. Fowler
Department Math & Computer Science, Clarkson University, Potsdam, NY, USA
E-mail address: `kfowler@clarkson.edu`

G.A. Gray
Sandia National Labs, Livermore, CA, USA
E-mail address: `gagray@sandia.gov`

T. Hemker
Department of Computer Science, TU Darmstadt, Darmstadt, Germany
E-mail address: `hemker@sim.tu-darmstadt.de`

M.D. Parno
Computation for Design and Optimization, MIT, Cambridge, MA, USA
E-mail address: `mparno@mit.come`