Deterministic Global Parameter Estimation for a Budding Yeast Model

T. D. Panning [*], **L.T. Watson**[*], **N. A. Allen** [*], **C. A. Shaffer**[*], **J. J. Tyson**[†]
Department of Computer Science[*],
Department of Biology[†],
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

Keywords—DIRECT (DIviding RECTangles) algorithm, direct search, MADS (Mesh Adaptive Direct Search) algorithm, computational biology

Abstract—Two parallel deterministic direct search algorithms are used to find improved parameters for a biological model. The model is a system of differential equations designed to simulate the cell cycle of budding yeast. Comparing the model simulation results to experimental data is difficult because most of the experimental data is qualitative rather than quantitative; an algorithm to convert simulation results to biological phenotypes is presented. Vectors of parameters defining the differential equation model are rated by a discontinuous objective function. Parallel results on a 2200 processor supercomputer are presented for a global optimization algorithm, DIRECT, and a local optimization algorithm, MADS.

1. INTRODUCTION

Molecular cell biology is ultimately about how cells convert genes into behavior. This includes how a cell creates proteins from genes, how those proteins interact, and how the proteins physically affect the cell. The central biological question addressed here is how the proteins interact with each other, and specifically, how those interactions regulate the cell cycle of budding yeast (*Saccharomyces cerevisiae*).

The budding yeast cell cycle consists of several phases, with cell division occurring in the final phase. A new cell starts in G1 phase, where it performs the normal cell functions while it waits until it reaches a size sufficient to replicate. Upon reaching the correct size, the cell enters S phase where it synthesizes a copy of its DNA. After DNA synthesis has completed, the cell enters M phase where the DNA copies are separated and the cell divides, creating two new cells that are in G1 phase.

To model the protein interactions that govern the cell cycle, biological modelers construct differential equations that describe the rate at which each protein concentration changes. The concentration of protein X is written as [X]. If protein A is degraded by protein B, then the model will include a differential equation similar to

$$\frac{d[\mathrm{A}]}{dt} = -c[\mathrm{B}],$$

with the initial condition $[\mathrm{A}](0) = A_0$. The parameter c determines the rate at which B degrades A. The budding yeast cell cycle model in [5] is composed of over 30 such differential equations, some of which are nonlinear. In addition, the budding yeast model consists of many different initial value problems, because some of the concentrations are reset when events (a), (b), (c), and (e) occur (as described in Section 2.1). In the above example, an additional initial value problem would be created if the value of [A] were reset to zero when [B] rose through (the value) one. This type of modeling can be aided by software tools such as the JigCell problem solving environment [1,2].

In the budding yeast model there are about 140 rate constant parameters similar to c in the above example. In some cases, these parameters can be calculated directly from laboratory experiments (e.g., protein half-lives), but most parameters would be difficult to obtain directly from experimentation. Normally, the modelers determine these remaining parameters by repeatedly making educated guesses, executing the model, comparing the simulation results with the laboratory data, and then refining their guesses. The biologists call this process "parameter twiddling" [2].

Parameter twiddling is a time intensive method; nonetheless, it was used to obtain a parameter vector for which the model's predictions are consistent with all but twelve of the 115 mutants of the budding yeast species for which experimental data exists. Obviously, the biologists would prefer a method that allows them to spend more time working on the model and less time twiddling parameters. In addition, a person can only keep track of a few parameters at one time, which

 ISBN 1-56555-304-7

makes it easy for a person to unwittingly miss a portion of the parameter space. For these reasons, the biologists would prefer to use a tool that could determine "good" parameters relatively quickly and accurately.

Section 3 describes one proposed mathematical formulation of "good" that may allow a computer code to find an acceptable vector of parameters. This formulation uses a discontinuous objective function that evaluates to zero when there is a perfect match between the experimental data and the simulation results, and it evaluates to increasingly larger numbers to indicate worse matches. Another possible formulation for future consideration would use a smooth system of inequalities that would be satisfied if and only if the simulation results are acceptable.

Section 2 describes the biological problem in some detail. Section 3 formulates a discontinuous objective function, reflecting biological criteria for an acceptable model. Two deterministic algorithms, DIRECT and MADS, that are applicable to global parameter estimation, are very briefly described in Section 4. Numerical results on a parallel supercomputer (2200 processor System X) are given in Section 5. Parallel efficiency and scalability are important issues to be addressed separately—the emphasis here is on the biological problem, the discontinuous objective function formulation, and the practical applicability of DIRECT and MADS to such optimization problems.

A *phenotype* is a description of the observable characteristics of an organism (as opposed to a *genotype*, which is a description of the genetic characteristics of an organism). Throughout this paper, the *observed* phenotype refers to the phenotype that was recorded in a laboratory experiment. The *predicted* phenotype refers to the phenotype that the mathematical model (with its associated parameters) predicts. The *wild type* is the normal strain of an organism. The *mutant* strains have genetic changes that make them behave differently from the wild type in some way.

2. PHENOTYPES

Experimental biologists have studied many budding yeast mutants to learn about the cell cycle regulatory system. Of these mutants, the modelers have chosen 115 that they wish to model. A model of budding yeast can be considered correct only if it is able to duplicate the behavior of these mutants. When the model is used to simulate a mutant, the parameter vector can be changed only in ways that are analogous to the genetic changes in the mutant. Consider the hypothetical proteins A and B presented in the previous section: if a mutant had a modified form of B that did not degrade A, then in the parameter vector for that mutant, c would be set to zero and all of the other parameters would be set to the wild type values.

When comparing the model to the experimental data, it is important to realize that much of the data from laboratory experiments is qualitative. Such data is of the form "the cell lived" or "the cell was unable to escape G1 phase." The quantitative data that is available (e.g., G1 phase length, cell mass at division) is generally imprecise. With all of these uncertainties, it is easy to suspect that many, clustered parameter vectors could allow the model to reproduce the experimental data. In fact, this is a desirable feature of the model; for survival, biological systems are necessarily insensitive to small fluctuations in the rate constants, so one would expect the model to behave similarly.

2.1 Rules of Viability

To compare the solutions of the differential equations with the experimental data, it is necessary to determine whether and when a simulated cell arrests. There are four rules of viability that determine whether a simulated cell is considered viable or inviable.

1. The modeled cell must execute the following events in order, or else the cell is considered inviable:
   (a) DNA licensed for replication (modeled by a drop in [Clb2] + [Clb5] below K_{ez2});
   (b) start of DNA synthesis (due to a subsequent rise in [Clb2] + [Clb5], causing [ORI] to increase above one) before a wild-type cell in the same medium would divide twice;
   (c) alignment of DNA copies (due to a rise in [Clb2], causing [SPN] to increase above one);
   (d) separation of DNA copies (modeled by [Esp1] increasing above 0.1, due to Pds1 proteolysis at anaphase);
   (e) cellular division (modeled by [Clb2] dropping below a threshold K_{ez}).
2. The cell is inviable if division occurs in an "unbudded cell" (i.e., if [BUD] does not reach the value 0.8 before event (e) occurs).
3. The cell cycle should be stable such that the squared relative difference of the masses and G1 phase lengths in the last two cycles is less than 0.05.
4. Lastly, the modeled cell is considered inviable if the cell mass is greater than four or less than one-fourth times the steady-state mass at division of the wild type in the same medium.

These rules are used by an algorithm (called a *transform*) that outputs a phenotype from the solutions

to the differential equations. The transform keeps track of what stage the cell is in, where the stages are demarcated by the events in the first rule of viability. The first stage is *unlicensed*, which ends when the first event, DNA licensed for replication, occurs. The other four stages are, in chronological order, *licensed*, *fired*, *aligned*, and *separated*. When the simulated cell is in the *separated* stage, cellular division signals the transition back to the *unlicensed* stage. If one of the rules of viability is broken, the transform sets an error flag and records the stage when the error occurred and the number of cycles (i.e., cell divisions) completed.

3. DISCONTINUOUS MINIMIZATION

The objective function takes the observed phenotype and predicted phenotype for all of the mutants and computes a nonnegative score. Zero indicates a perfect match and larger numbers indicate increasingly worse matches. The ensuing discussion uses the symbol O for observed phenotype values and P for predicted phenotype values.

A budding yeast phenotype for a single mutant is represented by a six-tuple $(v,\ g,\ m,\ a,\ t,\ c)$, where the viability $v \in \{\text{viable, inviable}\}$, the real number $g > 0$ is the steady state length of the G1 phase, the real number $m > 0$ is the steady state mass at division, the stage when arrest occurred is

$$a \in \{\text{unlicensed, licensed, fired, aligned, separated}\},$$

the positive integer t is the arrest type, and the nonnegative integer c is the number of successful cycles completed. The observed and predicted phenotypes are written $O = (O_v, O_g, O_m, O_a, O_t, O_c)$ and $P = (P_v, P_g, P_m, P_a, P_t, P_c)$, respectively. Arrest types cannot be compared unless the stage of arrest is the same for both phenotypes.

In what follows, the ωs and σs are constants defined in Table 1. The rating function, R, compares the observed and predicted phenotypes for a mutant. This rating function is a modified version of the one developed by N. Allen et al. [3]; the only difference is that if O_v or P_v is missing, then $R(O,P) = \omega_v$. The rating function is split into four cases depending on the viability of the observed and predicted phenotypes. If $O_v =$ inviable, $P_v =$ viable, and O_c is missing, then $R(O,P) = \omega_v$, the same as if $O_c = 0$. Otherwise, if a needed classifier is missing, the term is simply dropped and does not contribute to the objective function. In the case that classifiers are missing, this allows the objective function value to be at or near zero when viability is in agreement between the phenotypes, and forces larger objective function values when viability is not in agreement.

The rating function $R(O,P)$ when all classifiers are present is given by

$$\omega_g \times \left(\frac{O_g - P_g}{\sigma_g}\right)^2 + \omega_m \times \left(\frac{\ln \frac{O_m}{P_m}}{\sigma_m}\right)^2,$$

if $O_v =$ *viable* and $P_v =$ *viable*, by

$$\omega_v \times \frac{1}{1 + P_c},$$

if $O_v =$ *viable* and $P_v =$ *inviable*, by

$$\delta_{O,P} + \omega_c \times \left(\frac{O_c - P_c}{\sigma_c}\right)^2,$$

if $O_v =$ *inviable* and $P_v =$ *inviable*, and by

$$\omega_v \times \frac{1}{1 + O_c},$$

if $O_v =$ *inviable* and $P_v =$ *viable*, where δ is a real valued discrete function, used to assess a penalty for the arrest stage and type, given by

$$\delta_{O,P} = \begin{cases} \omega_a, & \text{if } O_a \neq P_a, \\ \omega_t, & \text{if } O_a = P_a \text{ and } O_t \neq P_t, \\ 0, & \text{if } O_a = P_a \text{ and } O_t = P_t. \end{cases}$$

The rating function is tuned by parameters to allow the modeler to adjust the relative importance of classifiers. The parameters given by Table 1 were set so that a rating of around ten indicates a critical error in the model's prediction of a phenotype.

Symbol	Definition	Value
ω_g	G1 length weight	1.0
σ_g	G1 length scale	10.0
ω_m	Mass at division weight	1.0
σ_m	Mass at division scale	$\ln 2$
ω_a	Arrest stage weight	10.0
ω_t	Arrest type weight	5.0
ω_c	Cycle count weight	10.0
σ_c	Cycle count scale	1.0
ω_v	Viability weight	40.0

TABLE 1. CONSTANTS USED IN OBJECTIVE FUNCTION.

Denote the real numbers by $\mathcal{R}$, the nonnegative integers $\{0, 1, 2, \ldots\}$ by $\mathcal{Z}_+$, and the integers by $\mathcal{Z}$. Let

$$\begin{aligned} \mathcal{P} &= (v, g, m, a, t, c) \\ &= \{\text{viable, inviable}\} \times (0, \infty)^2 \\ &\quad \times \{\text{unlicensed, licensed, fired, aligned, separated}\} \\ &\quad \times \{1, \ldots, 10\} \times \mathcal{Z}_+ \end{aligned}$$

be the space of all budding yeast phenotypes and let the domain of the objective function be the box

$$\Omega = \{x \in \mathcal{R}^{143} : s_i/u_i \le x_i \le s_i \times u_i, \\ i = 1, \ldots, 143\},$$

where $u \in \mathcal{R}^{143}$ are positive scale factors reflecting modelers' knowledge about the rate constants, and $s \in \mathcal{R}^{143}$ is the modeler's best guess point. Let $T_j : \Omega \to \mathcal{P}$ simulate the jth mutant with the parameters $x_1, \ldots, x_{143}$ and compute the phenotype. Then the objective function $f : \Omega \to [0, \infty)$ is defined by

$$f(x) = \sum_{j=1}^{N_m} \mu_j R(O_j, T_j(x)),$$

where N_m is the number of mutant experiments, and $\mu_i \in \{1, 4\}$ is a weight that indicates whether the ith mutant is of normal or high importance. The objective function value at the biologists' best previously known point [5] is 433.

4. ALGORITHMS

This section describes two algorithms that show promise for optimizing the discontinuous objective function described in the previous section. Consider the problem of minimizing $f : B \to \mathcal{R}$, where $B = [l, u] \subset \mathcal{R}^n$ is a box.

4.1. DIRECT

The DIRECT (Dividing Rectangles) global minimization algorithm [11] requires the objective function to be Lipschitz continuous to guarantee convergence. Even though the objective function used here is discontinuous, the DIRECT algorithm seems to be an efficient and reasonable deterministic sampling strategy worth trying.

The DIRECT algorithm is one of a class of deterministic direct search algorithms that does not require gradients. It works by iteratively dividing the search domain into boxes that have exactly one function value at the box's center. In each iteration, the algorithm determines which boxes are most likely to contain a better point than the current minimum point—these boxes are called "potentially optimal". It then subdivides the potentially optimal boxes along their longest dimensions. Intuitively, a box is considered potentially optimal if it has the potentially best function value for a given Lipschitz constant. The formal definition from [11] follows.

Definition. Suppose that the unit hypercube has been partitioned into m (hyper) boxes. Let c_i denote the center point of the ith box, and let d_i denote the distance from the center point to the vertices. Let $\epsilon > 0$ be a positive constant. A box j is said to be *potentially optimal* if there exists some $\tilde{K} > 0$ such that for all $i = 1, \ldots, m$,

$$f(c_j) - \tilde{K} d_j \le f(c_i) - \tilde{K} d_i, \qquad \text{for all } i = 1, \ldots, m,$$

$$f(c_j) - \tilde{K} d_j \le f_{\min} - \epsilon |f_{\min}|.$$

The DIRECT algorithm is described by the following six steps [7].

Step 1. Normalize the design space B to be the unit hypercube. Sample the center point c_i of this hypercube and evaluate $f(c_i)$. Initialize $f_{\min} = f(c_i)$, evaluation counter $m = 1$, and iteration counter $t = 0$.

Step 2. Identify the set S of potentially optimal boxes.

Step 3. Select any box $j \in S$.

Step 4. Divide the box j as follows:

(1) Identify the set I of dimensions with the maximum side length. Let δ equal one-third of this maximum side length.

(2) Sample the function at the points $c \pm \delta e_i$ for all $i \in I$, where c is the center of the box and e_i is the ith unit vector.

(3) Divide the box j containing c into thirds along the dimensions in I, starting with the dimension with the lowest value of $w_i = \min\{f(c + \delta e_i), f(c - \delta e_i)\}$, and continuing to the dimension with the highest w_i. Update $f_{\min}$ and m.

Step 5. Set $S = S - \{j\}$. If $S \neq 0$ go to Step 3.

Step 6. Set $t = t + 1$. If iteration limit or evaluation limit has been reached, stop. Otherwise, go to Step 2.

For an illustration of how the DIRECT algorithm searches the domain on an example problem, see [12]. Both serial [7] and parallel [8] versions of DIRECT have been described in the literature.

4.2. MADS

A MADS (Mesh Adaptive Direct Search) algorithm, as defined by Audet and Dennis [4], minimizes a nonsmooth function $f : \mathcal{R}^n \to \mathcal{R} \cup \{+\infty\}$ under general constraints $x \in \Omega \subseteq \mathcal{R}^n$, $\Omega \neq \emptyset$. If $\Omega \neq \mathcal{R}^n$, the algorithm works with f_Ω, which is equal to f on Ω and $+\infty$ outside Ω. Using f_Ω in lieu of f is called a "barrier" approach to handling arbitrary constraints $x \in \Omega$.

In each iteration, a MADS algorithm evaluates the objective function f_Ω at a finite number of trial points. Central to these algorithms is the concept of a mesh, which is a discrete set of points in $\mathcal{R}^n$. Every previous

trial point must lie on the current mesh, and in each iteration the algorithm may only generate new trial points on the current mesh. This is not as restrictive as it might sound because the algorithm changes the mesh after each iteration (with the restriction that all previously evaluated points remain in the new mesh).

To further define the mesh, three entities—Δ_k^m, D, S_k—must be introduced. First, the *mesh size parameter* $\Delta_k^m > 0$ controls the granularity of the mesh at iteration k; after the kth iteration, Δ_{k+1}^m is adjusted from Δ_k^m depending on the success of that iteration. The second entity is an $n \times n_D$ matrix D, where each column $D_{\cdot j} = Gz_j$ (for $j = 1, 2, \ldots, n_D$) for some fixed nonsingular generating matrix $G \in \mathcal{R}^{n\times n}$ and nonzero integer vector $z_j \in \mathcal{Z}^n$. The columns of D must also be a positive spanning set, $\mathrm{Pos}(D) = \mathcal{R}^n$ (i.e., the cone generated by nonnegative combinations of columns of D spans $\mathcal{R}^n$). Lastly, S_k is the set of points where the objective function has been evaluated by the start of iteration k. Now that those entities have been introduced, the current mesh can be precisely defined.

At iteration k, the current *mesh* is

$$M_k = \bigcup_{x\in S_k} \{x + \Delta_k^m Dz : z \in \mathcal{N}^{n_D}\}.$$

This ensures that all previously evaluated points are included in the mesh. It also shows that a smaller Δ_k^m will result in a more refined mesh, while a larger Δ_k^m will create a coarser mesh.

Now that the mesh has been defined, the iterations of a MADS algorithm can be described. Each iteration consists of two steps: the SEARCH step and the POLL step. The SEARCH step may evaluate f_Ω at any finite number of mesh points. At which mesh points f_Ω is evaluated depends on the precise MADS algorithm in use. A MADS algorithm may even do zero evaluations in the SEARCH step; the SEARCH step is said to be empty when no points are considered. If the SEARCH step fails to find a mesh point at which f_Ω is less than $\min_{x\in S_k} f_\Omega(x)$, then the algorithm performs the POLL step by generating and evaluating f_Ω at new trial points around the current incumbent solution x_k, where $f_\Omega(x_k) = \min_{x\in S_k} f_\Omega(x)$. The *poll size parameter* Δ_k^p limits the distance between x_k and the new trial points. The set of new trial points is called a *frame*, and x_k is called the *frame center*. The MADS frame is constructed using x_k, Δ_k^p, Δ_k^m, and D to obtain a set D_k of positive spanning directions.

Definition. At iteration k, the MADS *frame* is defined to be the set

$$P_k = \{x_k + \Delta_k^m d : d \in D_k\} \subset M_k,$$

where D_k is a positive spanning set such that $0 \notin D_k$ and for each $d \in D_k$,

- d can be written as a nonnegative integer combination of the columns of D: $d = Du$ for some vector $u \in \mathcal{N}^{n_D}$,
- the distance from the frame center x_k to a frame point $x_k + \Delta_k^m d \in P_k$ is bounded by a constant times the poll size parameter: $\Delta_k^m \|d\|_\infty \le \Delta_k^p \|D\|_\infty$ (where $\|\cdot\|_\infty$ indicates the maximum norm),
- limits (as defined in Coope and Price [6]) of the normalized sets D_k are positive spanning sets.

The algorithm evaluates f_Ω at points in the frame P_k until it encounters an improved point x^* ($f_\Omega(x^*) < f_\Omega(x_k)$) or it has evaluated f_Ω at all of the points in P_k.

After the algorithm has executed the SEARCH step and (conditionally) the POLL step, it sets the mesh size and poll size parameters, Δ_{k+1}^m and Δ_{k+1}^p, for the next iteration. If the iteration successfully found a better mesh point x_{k+1} such that $f_\Omega(x_{k+1}) < f_\Omega(x_k)$, then Δ_{k+1}^m will be larger than or equal to Δ_k^m; otherwise, Δ_{k+1}^m will be smaller than Δ_k^m. The poll size parameter Δ_{k+1}^p must be set such that $\Delta_{k+1}^m \le \Delta_{k+1}^p$, and it must satisfy

$$\liminf_{k\to\infty} \Delta_k^m = 0 \iff \liminf_{k\to\infty} \Delta_k^p = 0.$$

Exactly how Δ_{k+1}^m and Δ_{k+1}^p are generated is determined by the individual algorithm in use; see the example algorithm presented later in this section.

In summary, the MADS class of algorithms is described by the following five steps.

Step 1. Let $x_0 \in \Omega$ and $0 < \Delta_0^m \le \Delta_0^p$. Let D be an $n \times n_D$ matrix with the properties described earlier. Set the iteration counter $k := 0$.

Step 2. Perform the SEARCH step. This step varies among the individual algorithms; in all algorithms f_Ω is evaluated at a finite subset of points (called trial points) on the mesh M_k. If a trial point y is found such that $f_\Omega(y) < f_\Omega(x_k)$, then the algorithm may go to Step 4 with $x_{k+1} := y$.

Step 3. Perform the POLL step, evaluating f_Ω at points from the frame $P_k \subset M_k$ until a frame point x_{k+1} is found with $f_\Omega(x_{k+1}) < f_\Omega(x_k)$ or f_Ω has been evaluated at all of the points in P_k.

Step 4. Update Δ_{k+1}^m and Δ_{k+1}^p according to the specific algorithm's rules. In all algorithms,

(1) Δ_{k+1}^m is greater than or equal to Δ_k^m if an improved mesh point is found,

(2) Δ_{k+1}^m is less than Δ_k^m if an improved mesh point is not found,

(3) Δ_{k+1}^p is greater than or equal to Δ_{k+1}^m, and

(4) $\liminf_{j\to\infty} \Delta_j^m = 0 \iff \liminf_{j\to\infty} \Delta_j^p = 0$.

Step 5. If an appropriate stopping criterion has been met, stop. Otherwise, set $k := k + 1$ and go back to Step 2.

The previous discussion presents the MADS class of algorithms. The following discussion describes a specific instance of the class for $n = 2$. To emphasize the POLL step of the algorithm, there is no SEARCH step in the algorithm presented here.

In this MADS algorithm,

$$D = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -2 \end{pmatrix}.$$

Notice that a MADS mesh constructed using this matrix is identical to a mesh constructed using the matrix

$$B = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}.$$

However, $\|D\|_\infty = 2$ while $\|B\|_\infty = 1$; thus, a MADS frame constructed using D instead of B will extend twice as far in every direction. From D, the matrix D_k is generated (using random coefficients as described in [4]) at the beginning of the kth iteration so that it is a positive spanning set, and so that the (normalized) columns of D_i, for $i = 1, 2, \ldots$, are dense in the unit circle $\mathcal{S}^1$.

The mesh size parameter Δ^m is updated according to the rules:

$$\Delta_0^m = 1,$$

$$\Delta_{k+1}^m = \begin{cases} \Delta_k^m/4, & \text{if } x_k \text{ is a minimizing frame center,} \\ 4\Delta_k^m, & \text{if an improved mesh point is found, and if } \Delta_k^m \le \frac{1}{4}, \\ \Delta_k^m, & \text{otherwise.} \end{cases}$$

The poll size parameter Δ^p is updated according to the rule $\Delta_k^p = \sqrt{\Delta_k^m}$. These rules ensure that Δ_k^m is always a power of $1/4$ less than or equal to one, and Δ_k^m is always less than or equal to Δ_k^p.

5. RESULTS

All computation took place on System X, a cluster of 1100 dual-processor Mac G5 nodes.

NOMAD is a C++ implementation of the MADS class of algorithms. To take advantage of System X, NOMAD's implementation of the POLL step was parallelized using a master/worker paradigm. The master ran the MADS algorithm as presented and sent requests to the workers whenever objective function values were needed. NOMAD, started from the modeler's best point s, evaluated the objective function 135,000 times over 813 iterations using 128 processors, converging at a point for which the objective function value was 299 (this point correctly models all but ten of the mutants).

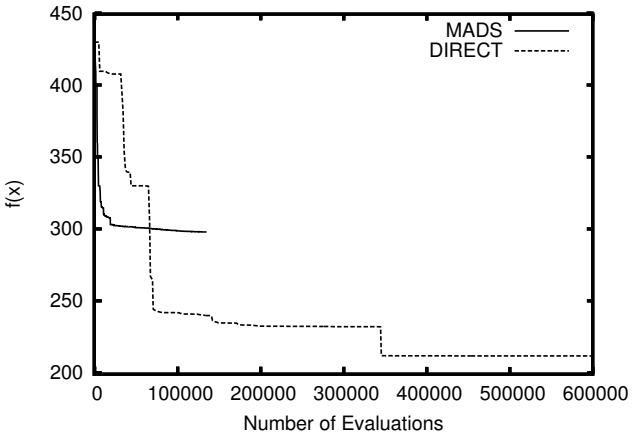

Figure 1. The objective function value at the best point found versus the number of evaluations for MADS and DIRECT.

pVTDirect [8] is a parallel implementation of DIRECT written in Fortran 95. While the DIRECT algorithm does not have a traditional "starting point", the first sample in each subdomain is always taken at the center of the subdomain bounding box. For this problem, the bounding box was designed so that the modeler's best point would be at the center and therefore would be evaluated before any other points. pVTDirect (with only one subdomain) ran for 473 iterations using 1024 processors and evaluated the objective function 1.5 million times, finding a point at which the objective function value was 212 (this point correctly models all but eight of the mutants).

Figure 1 shows the progress that each program was able to make in minimizing the objective function. While NOMAD was able to quickly find a better point than the modeler's best point, pVTDirect was eventually able to find an even lower point. This is expected behavior because NOMAD is designed for local optimization and pVTDirect is designed for global optimization, so NOMAD quickly found a nearby local minimum and stopped, but pVTDirect explored the parameter space and eventually found a better minimum. In a later run, NOMAD was started from pVTDirect's lowest point, but NOMAD was unable to make any further progress. After looking at Figure 1, it is tempting to believe that pVTDirect could have been stopped earlier (for instance, after 200,000 evaluations), and NOMAD started at pVTDirect's last best point could have found a point at which the objective function

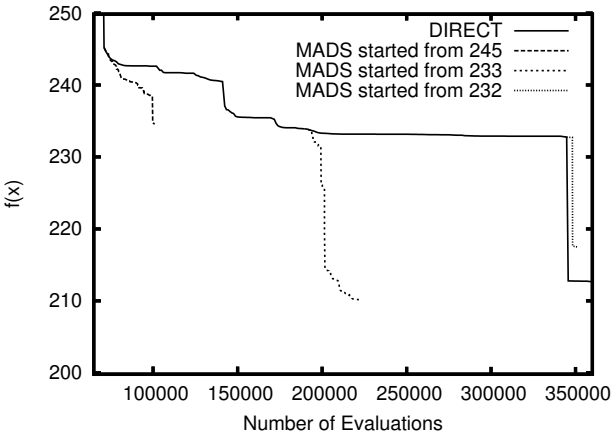

Figure 2. The performance of NOMAD when started from the best point at pVTDirect's 54th, 157th, and 239th iterations. The plots are shown as if the NOMAD runs started as soon as the respective pVTDirect iterations completed.

value was 212 or less. To test this, NOMAD was started at the best point at the 54th, 157th, and 239th iterations of pVTDirect. These points correspond to the beginning, middle, and end of the second-lowest plateau in Figure 1. As shown in Figure 2, NOMAD started from the middle point converged to a point at which the objective function value was 210. However, the NOMAD runs started at the beginning and end plateau points converge to worse points than pVTDirect's best point. These four extra NOMAD runs (including the one starting from pVTDirect's best point) show that an algorithm for improving intermediate results from pVTDirect is not so clear. Future work will explore heuristics for doing this.

ACKNOWLEDGMENTS

This work was partly supported by Defense Advanced Research Projects Agency (DARPA) grant F30602-02-0572.

REFERENCES

[1] N. A. Allen, C. A. Shaffer, M. T. Vass, N. Ramakrishnan, and L. T. Watson, "Improving the development process for eukaryotic cell cycle models with a modeling support environment", *Simulation*, 79 (2003) 674–688.

[2] N. Allen, L. Calzone, K. C. Chen, A. Ciliberto, N. Ramakrishnan, C. A. Shaffer, J. C. Sible, J. J. Tyson, M. Vass, L. T. Watson, and J. Zwolak, "Modeling regulatory networks at Virginia Tech", *OMICS*, 7 (2003) 285–299.

[3] N. A. Allen, K. C. Chen, J. J. Tyson, C. A. Shaffer, and L. T. Watson, "Computer evaluation of network dynamics models with application to cell cycle control in budding yeast", *IEE Systems Biology*, to appear.

[4] C. Audet and J. E. Dennis Jr., "Mesh adaptive direct search algorithms for constrained optimization", *SIAM J. Optim.*, to appear.

[5] K. C. Chen, L. Calzone, A. Csikasz-Nagy, F. R. Cross, B. Novak, and J. J. Tyson, "Integrative analysis of cell cycle control in budding yeast", *Molecular Biology of the Cell*, 15 (2004), 3841–3862.

[6] I. D. Coope and C. J. Price, "Frame-based methods for unconstrained optimization", *Journal of Optimization Theory and Applications*, 107 (2000) 261–274.

[7] J. He, L. T. Watson, N. Ramakrishnan, C. A. Shaffer, A. Verstak, J. Jiang, K. Bae, and W. H. Tranter, "Dynamic data structures for a direct search algorithm", *Comput. Optim. Appl.*, 23 (2002) 5–25.

[8] J. He, M. Sosonkina, C. A. Shaffer, J. J. Tyson, L. T. Watson, and J. W. Zwolak, "A hierarchical parallel scheme for a global search algorithm", in *Proc. High Performance Computing Symposium 2004*, J. Meyer (ed.), Soc. for Modeling and Simulation International, San Diego, CA, May, 2004, 43–50.

[9] J. He, M. Sosonkina, C. A. Shaffer, J. J. Tyson, L. T. Watson, and J. W. Zwolak, "A hierarchical parallel scheme for global parameter estimation in systems biology", in *Proc. 18th Internat. Parallel & Distributed Processing Symp.*, CD-ROM, IEEE Computer Soc., Los Alamitos, CA, 2004, 9 pages.

[10] J. He, M. Sosonkina, L. T. Watson, A. Verstak, and J. W. Zwolak, "Data-distributed parallelism with dynamic task allocation for a global search algorithm", in *Proc. High Performance Computing Symposium 2005*, M. Parashar and L. Watson (eds.), Soc. for Modeling and Simulation Internat., San Diego, CA, 2005, 164–172.

[11] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the Lipschitz constant", *J. Optim. Theory Appl.*, vol. 79, no. 1, (1993) 157–181.

[12] L. T. Watson and C. A. Baker, "A fully-distributed parallel global search algorithm", *Engineering Computations*, vol. 18, no. 1/2, 514–549.