J Glob Optim (2008) 40:719–738
DOI 10.1007/s10898-007-9273-7

Deterministic parallel global parameter estimation for a model of the budding yeast cell cycle

Thomas D. Panning · Layne T. Watson · Nicholas A. Allen · Katherine C. Chen · Clifford A. Shaffer · John J. Tyson

Received: 13 October 2006 / Accepted: 15 December 2007 / Published online: 12 February 2008

Abstract Two parallel deterministic direct search algorithms are combined to find improved parameters for 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 mutant phenotypes is presented. Vectors of the 143 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, a local optimization algorithm, MADS, and a hybrid of the two.

Keywords DIRECT (DIviding RECTangles) algorithm · Direct search · MADS (Mesh Adaptive Direct Search) algorithm · Computational biology

1 Introduction

Molecular cell biology describes how cells convert genes into behavior. This description includes how a cell creates proteins from genes, how those proteins interact, and how networks of interacting proteins determine physiological characteristics of the cell. The central biological question addressed here is how protein interactions regulate the cell cycle of budding yeast (*Saccharomyces cerevisiae*).

The budding yeast cell cycle [12–14] consists of four phases (G1, S, G2, M), with cell division occurring in the final phase. A newborn cell starts in G1 phase (unreplicated DNA), during which time it grows to a sufficiently large size to warrant a new round of DNA synthesis

T. D. Panning · N. A. Allen · K. C. Chen · C. A. Shaffer · J. J. Tyson
Departments of Computer Science and Biological Sciences, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA

L. T. Watson (✉)
Departments of Computer Science and Mathematics, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA
e-mail: ltw@ieee.org

(S phase). After DNA synthesis has completed, the cell passes briefly through G2 phase (replicated DNA) and then enters M phase (mitosis, where the two copies of each DNA molecule are separated and the cell divides, creating two new cells that are in G1 phase).

The protein interactions that govern these cell cycle events are modeled using differential equations that describe the rate at which each protein concentration changes. In general, the rate of change of the concentration of protein A is

$$\text{synthesis} - \text{degradation} - \text{binding} + \text{dissociation} - \text{inactivation} + \text{activation},$$

where "synthesis" is the rate at which new protein A molecules are synthesized from amino acids (which depends on the concentration of active messenger RNA molecules for a particular protein), "degradation" is the rate at which protein A is broken down into amino acids and polypeptide fragments (which depends on the activity of specific proteolytic enzymes), "binding" is the rate at which protein A combines with other molecules to form distinct molecular complexes, "dissociation" is the rate at which these complexes break apart, "inactivation" is the rate at which certain post-translational modifications (e.g., phosphorylation) of protein A are made, and "activation" is the rate at which these modifications are reversed (e.g., dephosphorylation). Each of these rates is itself a function of the concentrations of the interacting species in the network. For example,

$$\begin{aligned}
\text{synthesis} &= k_1[\text{transcription factor}],\\
\text{degradation} &= k_2[\text{proteolytic enzyme}][\text{A}],\\
\text{binding} &= k_3[\text{A}][\text{B}], \text{ where B is a binding partner},\\
\text{dissociation} &= k_4[\text{AB}],\\
\text{inactivation} &= \frac{k_5[\text{kinase}][\text{A}]}{J_5 + [A]},\\
\text{activation} &= \frac{k_6[\text{phosphatase}][\text{A}_p]}{J_6 + [\text{A}_p]}, \quad \text{where } \text{A}_p \text{is the phosphorylated form of A.}
\end{aligned}$$

In these rate laws, [A] is the concentration of protein A, [AB] is the concentration of the compound that is created when proteins A and B bind together, [A][B] is the concentration of A times the concentration of B, the ks are rate constants, and the Js are Michaelis constants. Other differential equations must be used to determine the temporal dynamics of the concentrations of the "transcription factor," "proteolytic enzyme," "kinase," etc.

The budding yeast cell cycle model consists of 36 such differential equations for two classes of variables: regulatory proteins and physiological "flags." The regulatory proteins are triggers for specific events of the budding yeast cell cycle: Cln2 triggers budding, Clb5 triggers DNA synthesis, Clb2 drives cells into mitosis, and Esp1 drives cells out of mitosis and back to G1. The physiological "flags" are dummy variables that track the strength of these trigger proteins. For example, "BUD" is an integral of the activity of Cln2; when BUD $= 1$, a new bud is initiated. "ORI," an integral of [Clb5], represents the state of "origins of replication." When ORI $= 1$ ("fired" origins), DNA synthesis is initiated; at cell division, when [Clb2]$+$[Clb5] drops below a threshold level, ORI is reset to zero ("licensed" origins). Finally, "SPN" represents the alignment of replicated chromosomes on the mitotic spindle. SPN is driven by Clb2 activity; i.e., SPN is an integral of [Clb2].

In the budding yeast model there are 143 rate constant parameters (ks, Js, etc.). In some cases, these parameters can be calculated directly from laboratory experiments (e.g., apparent protein half-lives), but most parameters are difficult to obtain directly from experimentation. Normally, modelers determine the remaining parameters by making educated guesses,

solving the differential equations numerically, comparing the simulation results with laboratory data, and then refining their guesses. (Modelers call this process "parameter twiddling" [1].) For the budding yeast cell cycle, the laboratory data consists of observed phenotypes of more than 100 mutant yeast strains constructed by disabling and/or over-expressing the genes that encode the proteins of the regulatory network.

Although parameter twiddling is extremely tedious, it was used to obtain a parameter vector $(s_1, s_2, \ldots, s_{143})$ for which the model's predictions are consistent with almost all of the budding yeast mutants being modeled. Obviously, the modelers 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 makes it easy for him or her to unwittingly miss a portion of the parameter space. For these reasons, modelers would prefer to use a tool that determines "good" parameters automatically, quickly and accurately.

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 briefly described in Sect. 4. Numerical results on the supercomputer System X (a 2200 processor cluster) are given in Sect. 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.

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 Observed and predicted phenotypes

Experimental biologists have studied many budding yeast mutants to learn about the cell cycle regulatory system. Of these mutants, 115 were chosen to model (see Appendix A). A model of budding yeast can be considered acceptable only if it is able to duplicate the behavior of most of these mutants. (It would be too much to expect a model to account for *all* the "observations" because of lingering uncertainties about the reaction network and inevitable mistakes in phenotyping mutants.) When the model is used to simulate a mutant, the parameter vector can be changed only in ways that are dictated by 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 bind to A, then in the parameter vector for that mutant, k_3 would be set to zero and all the other parameters would be kept at 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 is viable but considerably larger than wild type cells" or "the cell arrests in G1 phase and eventually dies." The quantitative data that is available (e.g., duration of G1 phase, cell mass at division) is generally imprecise. With all these uncertainties, there may be many, clustered parameter vectors that allow the model to reproduce the experimental data sufficiently well.

2.1 Rules of viability

To compare solutions of the differential equations with experimental data, it is necessary to predict cell cycle properties from a simulation of regulatory protein dynamics. Viability is determined by four rules:

1. The modeled cell must execute the following events in order, or else the modeled 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, i.e., the squared relative differences of the masses and G1 phase durations in the last two cycles should both be less than 0.05.
4. Lastly, the modeled cell is considered inviable if cell mass at division is greater than four times or less than one-fourth times the steady-state mass at division of the wild type in the same medium.

As mentioned in Sect. 1, the physiological flags are reset when certain events (a–e) occur. For a complete description of the resetting rules, see [5].

The viability rules are used by an algorithm [2,3] (called a *transform*) that outputs a phenotype from a solution of the differential equations. The transform keeps track of what stage the cell is in, where the stages are demarcated by the events (a–e) above. The first stage is *unlicensed*, which ends when the first event, origin relicensing, 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. The relations among the stages, events, and biological phases of the cell are shown in Fig. 1. 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 from the time when the mutation was expressed to the time when the cell arrested.

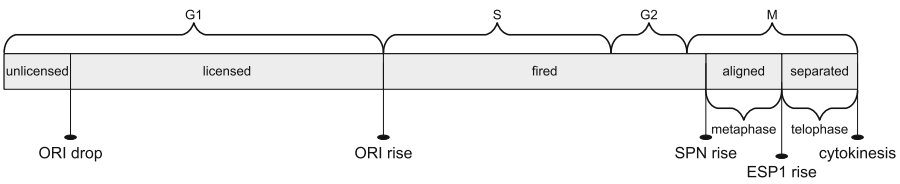

Fig. 1 The five stages of the cell cycle, delineated by the events described in the first rule of viability. The four biological phases of the cell are above the stages, and two of the phases within M phase are shown below their corresponding stages

2.2 Initial conditions

In the experimental data set, many of the mutations are conditional, that is, the mutant cells when grown under "normal" conditions (say, glucose medium at room temperature) behave like wild-type cells, but when grown under "restrictive" conditions (say, galactose medium or elevated temperature) the cells express the genetic mutation and the aberrant phenotype. To model this situation at sample points in parameter space, start a "wild-type" simulation from arbitrary (but reasonable) initial conditions and integrate the differential equations for two full cycles, in order to wash out any effects of the initial conditions. Then record the state of the control system just after origin relicensing (see Fig. 1) at the beginning of the third cycle. These recorded values are used as initial conditions for simulating a steady state wild-type cell and for simulating each of the mutants.

3 Formulation as a discontinuous minimization problem

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 in minutes, the real number $m > 0$ is the steady state mass at division expressed as a multiple of the wild type's steady state mass at division in the same medium (e.g., glucose or galactose), the stage when arrest occurs is

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

the positive integer t is the arrest type (e.g., if events occur in improper order), and the nonnegative integer c is the number of successful cycles completed. Arrest types cannot be compared unless the stage of arrest is the same for both phenotypes. 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. Then

$$\mathcal{P} = \{(v, g, m, a, t, c)\} = \{\text{viable, inviable}\} \times (0, \infty)^2 \times A \times \{1, \ldots, 10\} \times \mathcal{Z}_+$$

is the space of all budding yeast phenotypes, where $\mathcal{Z}_+$ denotes the nonnegative integers $\{0, 1, 2, \ldots\}$.

The rating function, R, computes a non-negative real number that expresses the deviations between the observed and predicted phenotypes for a mutant. This rating function is a modified version of the one developed by N. Allen et al. [2]; the only difference is that if P_v is missing (if integration fails for some reason), 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 $O_v = \text{inviable}$, $P_v = \text{viable}$, 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.

Table 1 Constants used in the objective function

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

In what follows, the ωs and σs are constants defined in Table 1. The rating function when all classifiers are present is given by

$$R(O,P)=\begin{cases}\omega_g\times\left(\frac{O_g-P_g}{\sigma_g}\right)^2+\omega_m\times\left(\frac{\ln\frac{O_m}{P_m}}{\sigma_m}\right)^2, & \text{if } O_v=viable \text{ and } P_v=viable,\\ \omega_v\times\frac{1}{1+P_c}, & \text{if } O_v=viable \text{ and } P_v=inviable,\\ \delta_{O,P}+\omega_c\times\left(\frac{O_c-P_c}{\sigma_c}\right)^2, & \text{if } O_v=inviable \text{ and } P_v=inviable,\\ \omega_v\times\frac{1}{1+O_c}, & \text{if } O_v=inviable \text{ and } P_v=viable,\end{cases}$$

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 that allow adjusting 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.

Denote the real numbers by $\mathcal{R}$, the integers by $\mathcal{Z}$, the number of parameters by N_p, and the number of mutants by N_m. Let the domain of the objective function be the box

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

where $u\in\mathcal{R}^{N_p}$ is a vector of positive scale factors (most components of u are approximately 100) reflecting a priori limits on the rate constants, and $s\in\mathcal{R}^{N_p}$ is the modeler's best guess point. For any $x\in\Omega$, the simulated phenotype of mutant $j\in\{1,2,\ldots,N_m\}$ is $P_j(x)\in\mathcal{P}$. The objective function is the weighted sum of the rating function over all of the mutants, that is,

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

where $\mu_i>0$ indicates the relative importance of the ith mutant. The objective function value at the best previously known point [5] is 470.

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 $f : \Omega \to \mathcal{R}$, where $\Omega = [l, u] \subset \mathcal{R}^n$ 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 1 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 $\epsilon \geq 0$ be a positive constant. A box j is said to be *potentially optimal* if there exists some $\tilde{K} > 0$ such that for all $i = 1, \ldots, m$,

$$f(c_j) - \tilde{K} d_j \leq f(c_i) - \tilde{K} d_i, \qquad \text{for all } i = 1, \ldots, m,$$
$$f(c_j) - \tilde{K} d_j \leq f_{\min} - \epsilon |f_{\min}|.$$

The DIRECT algorithm is described by the following six steps [10].

Step 1. Normalize the design space Ω 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 Fig. 2. Both serial [10] and parallel [7–9] versions of DIRECT have been described in the literature.

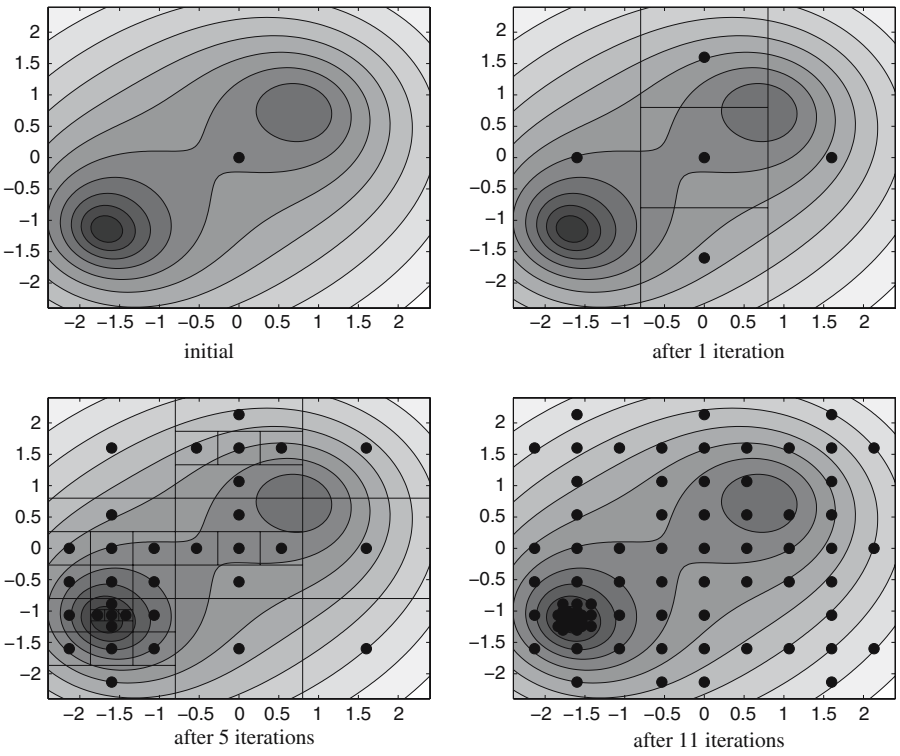

Fig. 2 These graphs show the function evaluations that DIRECT performed after zero, one, five, and eleven iterations. Comparing the first and second graphs shows how DIRECT divides a two-dimensional box. The second and third graphs include the rectangles that DIRECT had created. After five iterations, DIRECT has found the global minimum at $(-1.8, -1.1)$. After the fifth iteration DIRECT has explored the domain, subdividing most of the larger boxes. After eleven iterations, DIRECT has evaluated the function at points near the local minimum

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 $D_j = Gz_j$ (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, $\text{Pos}(D) = \mathcal{R}^n$ (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.

Definition 2 At iteration k, the current *mesh* is defined to be

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

This definition 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 3 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: $\Delta_k^m \|d\|_\infty \le \Delta_k^p \|D\|_\infty$,
- 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. A typical algorithm uses the following rules to set the mesh size parameter: $\Delta_0^m = 1$,

and

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

The same example algorithm then uses the simple rule $\Delta_k^p = \sqrt{\Delta_k^m}$ to determine the value of Δ_{k+1}^p. 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.

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

Step 1. Let $x_0 \in \Omega$ and $0 < \Delta_0^m \leq \Delta_0^p$. 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) $\liminf_{j\to\infty} \Delta_j^m = 0$ if and only if $\liminf_{j\to\infty} \Delta_j^p = 0$.

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, and Fig. 3 shows how that algorithm behaves on an example problem. 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

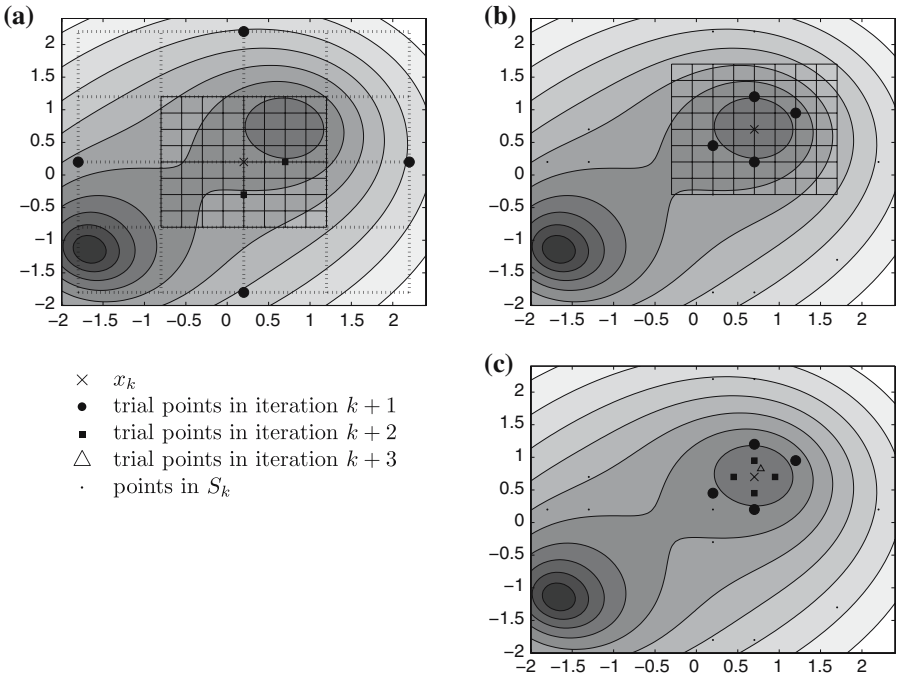

Fig. 3 These three graphs show how a MADS algorithm can refine the mesh, choose different poll directions, and contract the search area. (**a**) the intersections of the dotted lines indicate points that met the criteria for frame points in the first iteration. From these possibilities, the algorithm chose the four points indicated by the large circles; these four points constitute P_1. At all of these points, the function is higher than at x_0, so the algorithm refined the mesh by setting $\Delta_2^m = \Delta_1^m/4$. The intersections of the solid lines in the same graph indicate possible frame points for the second iteration. The algorithm evaluated the function at only two points in P_2 because the function is lower at the second point than at x_0, (**b**) shows how the mesh allows a MADS algorithm to choose different poll directions at each iteration, (**c**) highlights three consecutive iterations of the algorithm. In the first two iterations, the algorithm is unable to find an improved mesh point, therefore it restricts the search area to be closer to x_k. This can be seen by looking at the points evaluated in the three iterations; the circles are the furthest away from x_k, the squares are closer, and the triangle is the closest. There is only one triangle because the function value at that point is lower than $f(x_k)$, so the algorithm stopped the POLL step and went on to the next iteration

in the unit circle $\mathcal{S}^1$. The mesh size parameter Δ^m and poll size parameter Δ^p are updated using the same rules as presented earlier.

The example function in Fig. 3 has two local minima in the chosen domain. The basin of attraction for the global minimum point is smaller than that of the other local minimum point. For the example problem, $\Omega = \{(x_1, x_2) : x_1 \in (-2, 2.4), x_2 \in (-2, 2.4)\}$.

5 Results

Because it takes approximately 17 s to evaluate the objective function on a 2.3 GHz PowerPC G5 processor, parallel versions of both algorithms were used. All computation took place on a cluster of 1100 dual-processor Mac G5 nodes, called System X, at Virginia Tech.

NOMAD [4] is a C++ implementation of the MADS class of algorithms. The specific MADS algorithm used here is the same as the example algorithm given in Sect. 4, with

$\Delta_0^m = 1/4$. 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 in Sect. 4 and sent requests to the workers whenever objective function values were needed. NOMAD, started from the modeler's best point, 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.

pVTDirect [7] 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 [5] would be at the center and therefore would be evaluated before any other points. pVTDirect (with only one subdomain and $\epsilon = 0$) 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.

The first set of runs (the results of which are illustrated in Figs. 4 and 5) use the same initial conditions simulating all of the points, not different initial conditions as described in Sect. 2.2. (These runs led to the discovery of the necessity of the procedure in 2.2.) Figure 4 shows the progress that each algorithm 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 eventually found an even lower point. In a later run, NOMAD was started from pVTDirect's lowest point, but NOMAD was unable to make any further progress. After looking at Fig. 4, 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 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 Fig. 4. As shown in Fig. 5, 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 using NOMAD to get more value out of a pVTDirect run is not simple.

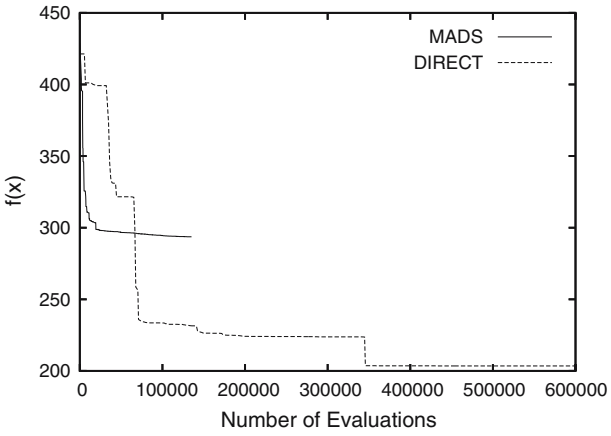

Fig. 4 The objective function value at the best point found versus the number of evaluations for MADS and DIRECT (The computations on which Figs. 4 and 5 are based used a standard set of initial conditions for every simulation, not the more accurate updating of initial conditions described in Sect. 2.2.)

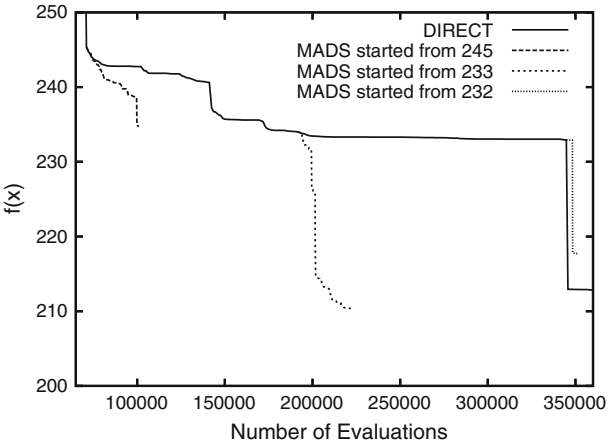

Fig. 5 The performance of NOMAD when started from the best point at pVTDirect's 54th, 157th, and 239th iterations. The plots are shown as if the NOMAD runs started as soon as the respective pVTDirect iterations completed

Table 2 Results for the runs that started from the best known point[5]

Run	lowest f	# evaluations	# CPUs	CPU hours
MADS	325	77,221	64	384
DIRECT, $\epsilon = 0$	233	1,243,429	400	8,895
DIRECT, $\epsilon = 0.1$	365	1,452,597	400	10,703

When the initial conditions were generated (as described in Sect. 2.2) for the best points from the previously described runs, the points received considerably worse objective function values, leading to the realization that the initial conditions have to be chosen per Sect. 2.2 for each parameter vector. To remedy this, the objective function was modified to generate proper initial conditions (see Sect. 2.2) for each point before evaluating the point. NOMAD and pVTDirect were then rerun on the corrected objective function. For this set of runs, pVTDirect was run twice, once with $\epsilon = 0$ and again with $\epsilon = 0.1$. Table 2 gives basic information for each method, and the results of these three runs are shown in Fig. 6.

Figures 7, 8, and 9 give an idea of what areas of parameter space the different runs explored. The distance measure used for these figures was $\sum_{i=1}^{N_p} |s_i - p_i|/s_i$, where s is the modeler's best parameter vector and p is the parameter vector of the point being plotted. These figures confirm that DIRECT evaluates points further away from the starting point than MADS. What is not shown in Figs. 8 and 9 is that DIRECT found only 77,752 points that evaluated to less than 480 when ϵ was set to 0.1, but it found 565,982 such points when ϵ was set to 0, even though both runs evaluated approximately the same number of points. This, combined with the information shown in Figs. 8 and 9, shows that setting ϵ to 0.1 caused DIRECT to spend more time dividing large boxes and less time refining small boxes.

However, Fig. 5 showed that MADS may be able to quickly improve on the best points that DIRECT has found. Unfortunately, it is difficult to guess which points from DIRECT will be good starting points for MADS. To find points that scored well but were reasonably far apart,

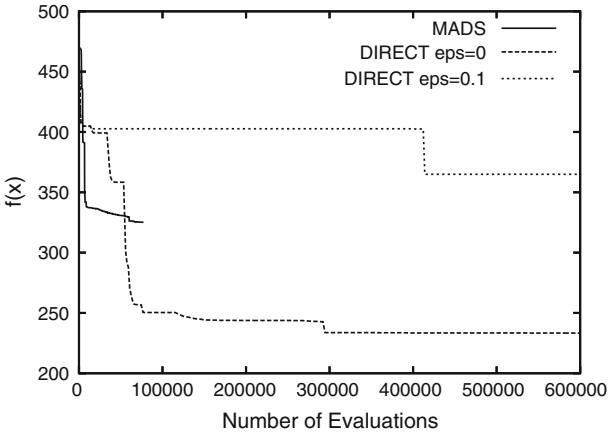

Fig. 6 The objective function value at the best point found so far versus the number of evaluations for MADS, DIRECT with $\epsilon = 0$, and DIRECT with $\epsilon = 0.1$

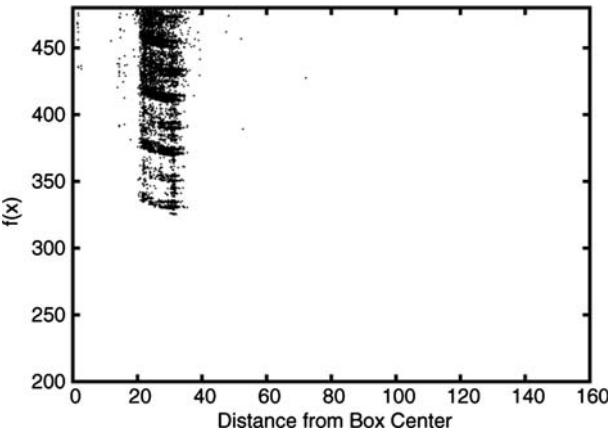

Fig. 7 The distribution of points that evaluated to less than 480 when MADS ran

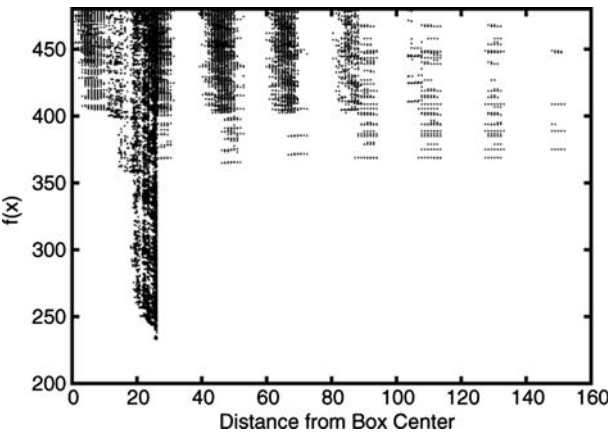

Fig. 8 The distribution of points that evaluated to less than 480 when DIRECT ran with $\epsilon = 0$

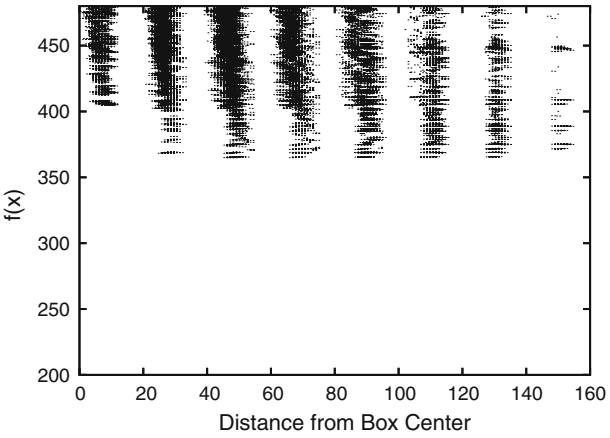

Fig. 9 The distribution of points that evaluated to less than 480 when DIRECT ran with $\epsilon = 0.1$

the following algorithm was used. Define the closed ball $B(x, \delta) = \{y \in \mathcal{R}^n | \|y - x\| \leq \delta\}$, and denote the complement of the set S by $\mathcal{C}(S)$.

Step 1. Let P be all of the points that DIRECT evaluated when ϵ was set to 0. Set $k := 1$.

Step 2. Find $s_k \in P$ such that $f(s_k) \leq f(p)$ for all $p \in P \cap \mathcal{C}\left(\bigcup_{i=1}^{k-1} B(s_i, 2)\right)$.

Step 3. If $f(s_k) \geq 300$, then stop, else set $k := k + 1$ and go to Step 2.

This algorithm yielded 13 points to use as starting points for MADS runs. The results of those runs are given in Table 3. In this table, there does not seem to be any relationship between $f(s_i)$ and the objective function value at the point to which MADS converges. It is also interesting that for 12 of the runs, MADS converges to a point that evaluates to 220–240, but the third run converges to a point that evaluates to a much lower score of 189. This is the lowest score found by any of the runs.

6 Conclusion

Even with a discontinuous objective function and a 143-dimension search domain, both DIRECT and MADS performed well. When ϵ was set to zero, DIRECT explored the parameter space and refined the boxes near local minima. When ϵ was set to 0.1, DIRECT used most of its evaluations to explore the parameter space. Any inferences about the choice of ϵ must be made with caution, since the box center here was already a very good point, and sizeable volumes of the 143-dimensional space still remain unsampled. MADS was almost always able to find a better point than its starting point, and it did so with far fewer evaluations than DIRECT. Using the two algorithms together yielded the lowest-scoring point.

The best previously known parameter vector from Chen et al. [5] has an objective function value of 470, and correctly models all of the mutants except (numbers from Appendix A) 7, 16, 35, 41, 45, 53, 76, 93, 97, 103, 104, and 110. The best point from DIRECT/MADS has an objective function value of 189, and correctly models all but mutants 35, 41, 45, 53, 76, 93, and 110.

How sensitive is the biological model (as reflected in the objective function) to local disturbances of these parameter vectors? If the DIRECT/MADS parameter values were rounded

Table 3 The MADS runs that started from points found during the DIRECT (with $\epsilon = 0$) run. All MADS runs used 64 processors

i	$f(s_i)$	final f	# evaluations	CPU hours
1	233	233	1,590	12
2	244	234	22,153	105
3	244	189	76,978	384
4	249	226	39,990	189
5	250	230	82,551	384
6	250	231	79,869	385
7	255	224	59,884	266
8	257	220	80,361	385
9	260	228	86,251	384
10	262	233	81,431	385
11	291	228	79,110	384
12	292	244	53,257	238
13	293	238	84,035	384

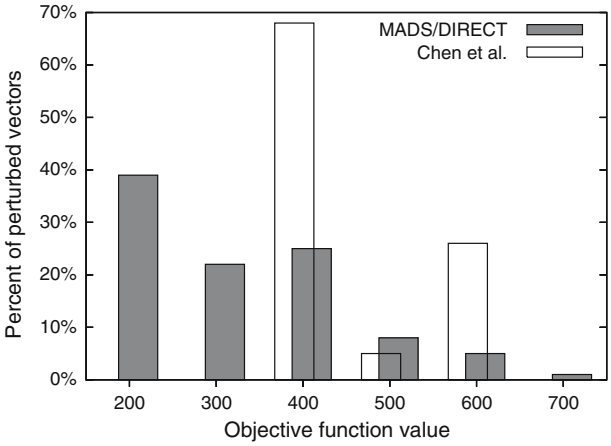

Fig. 10 The best DIRECT/MADS parameter vector and the parameter vector from Chen et al. [5] were randomly perturbed by up to $\pm 0.5\%$ in all dimensions, and the objective function was evaluated at each of the perturbed vectors. This figure shows the percentage of vectors that scored in $[x, x + 100)$, for $x = 200, 300, \ldots, 700$

to two significant figures, would the model be such a good fit to the data? To investigate this question, random perturbations (up to $\pm 0.5\%$) were applied to all of the parameter values in both the Chen et al. [5] parameter vector and the best DIRECT/MADS parameter vector, and histograms of the resulting objective function values were computed (Fig. 10). For the Chen et al. [5] parameter vector, about 25% of perturbations (white bars in Fig. 10) give significantly worse fits to the data, a reflection of the fact that behavior of the model is quite sensitive to a small number of the parameters, as described in detail in [5]. The DIRECT/MADS parameter vector (grey bars in Fig. 10) appears to be not only better but also its perturbations are less likely to produce inferior models than for the Chen et al. parameter vector. Thus the

combination of DIRECT and MADS to perform parameter optimization on discontinuous objective functions in very high dimensional (>100) parameter spaces is not only feasible computationally but also can find "good" parameter vectors that improve on the best estimates of expert modelers.

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

Appendix I: Mutant phenotypes

Listed below are the phenotypic characteristics of all the mutants used to construct the objective function. The data are expressed as a six-tuple (v, g, m, a, t, c) as described in Sect. 3. The meanings of the values for t are described in [2]. For all fields other than viability, a dash (–) means that the data is either not available or not applicable for that mutant

Mutant name	v	g	m	a	t	c
1. Wild type in glucose	Viable	35.2	1	–	–	–
2. Wild type in galactose	Viable	109	1	–	–	–
3. *cln1*Δ *cln2*Δ	Viable	–	2	–	–	–
4. *GAL-CLN2 cln1*Δ *cln2*Δ	Viable	–	0.5	–	–	–
5. *cln1*Δ *cln2*Δ *sic1*Δ	Viable	–	–	–	–	–
6. *cln1*Δ *cln2*Δ *cdh1*Δ	Viable	–	–	–	–	–
7. *GAL-CLN2 cln1*Δ *cln2*Δ *cdh1*Δ	Viable	–	1.7	–	–	–
8. *cln3*Δ	Viable	–	1.7	–	–	–
9. *GAL-CLN3*	Viable	–	0.44	–	–	–
10. *bck2*Δ	Viable	–	1.4	–	–	–
11. Multi-copy *BCK2*	Viable	–	0.8	–	–	–
12. *cln1*Δ *cln2*Δ *bck2*Δ	Viable	–	1.7	–	–	–
13. *cln3*Δ *bck2*Δ	Inviable	–	–	Licensed	5	0
14. *cln3*Δ *bck2*Δ *GAL-CLN2 cln1*Δ *cln2*Δ	Viable	–	–	–	–	–
15. *cln3*Δ *bck2*Δ multi-copy *CLN2*	Inviable	–	–	Licensed	5	–
16. *cln3*Δ *bck2*Δ *GAL-CLB5*	Inviable	–	–	–	–	–
17. *cln3*Δ *bck2*Δ *sic1*Δ	Inviable	–	–	–	–	–
18. *cln1*Δ *cln2*Δ *cln3*Δ	Inviable	–	–	Licensed	5	0
19. *cln1*Δ *cln2*Δ *cln3*Δ *GAL-CLN2*	Viable	–	–	–	–	–
20. *cln1*Δ *cln2*Δ *cln3*Δ *GAL-CLN3*	Viable	–	–	–	–	–
21. *cln1*Δ *cln2*Δ *cln3*Δ *sic1*Δ	Viable	10	3.5	–	–	–
22. *cln1*Δ *cln2*Δ *cln3*Δ *cdh1*Δ	Inviable	–	–	Separated	3	–
23. *cln1*Δ *cln2*Δ *cln3*Δ multi-copy *CLB5*	Viable	–	–	–	–	–
24. *cln1*Δ *cln2*Δ *cln3*Δ *GAL-CLB5*	Viable	–	–	–	–	–
25. *cln1*Δ *cln2*Δ *cln3*Δ multi-copy *BCK2*	Viable	–	–	–	–	–
26. *cln1*Δ *cln2*Δ *cln3*Δ *GAL-CLB2*	Inviable	–	–	Licensed	5	0
27. *cln1*Δ *cln2*Δ *cln3*Δ *apc-ts*	Inviable	–	–	Aligned	3	0
28. *sic1*Δ	Viable	15	1	–	–	–
29. *GAL-SIC1*	Viable	135	2	–	–	–
30. *GAL-SIC1-db*Δ	Inviable	–	–	Licensed	5	0
31. *GAL-SIC1 cln1*Δ *cln2*Δ	Inviable	–	–	Licensed	5	–
32. *GAL-SIC1 cln1*Δ *cln2*Δ *cdh1*Δ	Inviable	–	–	Licensed	5	–

Mutant name	*v*	*g*	*m*	*a*	*t*	*c*
33. *GAL-SIC1 GAL-CLN2 cln1*Δ *cln2*Δ	Viable	–	–	–	–	–
34. *GAL-SIC1 GAL-CLN2 cln1*Δ *cln2*Δ *cdh1*Δ	Viable	–	–	–	–	–
35. *sic1*Δ *cdh1*Δ	Inviable	–	–	Unlicensed	1	1
36. *sic1*Δ *cdh1*Δ *GALL-CDC20*	Viable	–	–	–	–	–
37. *cdh1*Δ	Viable	–	0.6	–	–	–
38. Cdh1 constitutively active	Inviable	–	–	Fired	3	–
39. *cdc6*Δ*2-49*	Viable	–	–	–	–	–
40. *sic1*Δ *cdc6*Δ*2-49*	Viable	–	–	–	–	–
41. *cdh1*Δ *cdc6*Δ*2-49*	Viable	20	2.0	–	–	–
42. *clb1*Δ *clb2*Δ	Inviable	–	–	Fired	3	0
43. *GAL-CLB2*	Viable	–	–	–	–	–
44. Multicopy *GAL-CLB2*	Inviable	–	–	Separated	3	0
45. *GAL-CLB2 sic1*Δ	Inviable	–	–	Separated	3	0
46. *GAL-CLB2 cdh1*Δ	Inviable	–	–	–	–	–
47. *CLB2-db*Δ	Inviable	–	–	Separated	3	–
48. *CLB2-db*Δ in galactose	Inviable	–	–	Separated	3	–
49. *CLB2-db*Δ multicopy *SIC1*	Viable	–	–	–	–	–
50. *CLB2-db*Δ *GAL-SIC1*	Viable	–	–	–	–	–
51. *CLB2-db*Δ *clb5*Δ	Inviable	–	–	Separated	3	0
52. *CLB2-db*Δ *clb5*Δ in galactose	Viable	–	–	–	–	–
53. *GAL-CLB2-db*Δ	Inviable	–	–	Separated	3	–
54. *clb5*Δ *clb6*Δ	Viable	65	–	–	–	–
55. *cln1*Δ *cln2*Δ *clb5*Δ *clb6*Δ	Inviable	–	–	Licensed	5	0
56. *GAL-CLB5*	Viable	–	–	–	–	–
57. *GAL-CLB5 sic1*Δ	Inviable	–	–	Unlicensed	–	1
58. *GAL-CLB5 cdh1*Δ	Inviable	–	–	–	–	–
59. *CLB5-db*Δ	Viable	–	–	–	–	–
60. *CLB5-db*Δ *sic1*Δ	Inviable	–	–	–	–	1
61. *CLB5-db*Δ *pds1*Δ	Viable	–	–	–	–	–
62. *CLB5-db*Δ *pds1*Δ *cdc20*Δ	Inviable	–	–	Separated	3	0
63. *GAL-CLB5-db*Δ	Inviable	–	–	–	–	1
64. *cdc20-ts*	Inviable	–	–	Aligned	3	0
65. *cdc20*Δ *clb5*Δ	Inviable	–	–	Aligned	3	0
66. *cdc20*Δ *pds1*Δ	Inviable	–	–	Separated	3	0
67. *cdc20*Δ *pds1*Δ *clb5*Δ	Viable	–	–	–	–	–
68. *GAL-CDC20*	Inviable	–	–	Fired	10	0
69. *cdc20-ts mad2*Δ	Inviable	–	–	Aligned	3	0
70. *cdc20-ts bub2*Δ	Inviable	–	–	Aligned	3	0
71. *pds1*Δ	Viable	–	–	–	–	–
72. *esp1-ts*	Inviable	–	–	Aligned	1	0
73. *PDS1-db*Δ	Inviable	–	–	Aligned	1	0
74. *GAL-PDS1-db*Δ	Inviable	–	–	Aligned	1	0
75. *GAL-PDS1-db*Δ *esp1-ts*	Inviable	–	–	Aligned	1	0
76. *GAL-ESP1 cdc20-ts*	Inviable	–	–	Separated	3	0
77. *tem1*Δ	Inviable	–	–	Separated	3	0
78. *GAL-TEM1*	Viable	–	–	–	–	–

Mutant name	*v*	*g*	*m*	*a*	*t*	*c*
79. *tem1-ts GAL-CDC15*	Viable	–	–	–	–	–
80. *tem1*Δ *net1-ts*	Viable	–	–	–	–	–
81. *tem1-ts* multicopy *CDC14*	Viable	–	–	–	–	–
82. *cdc15*Δ	Inviable	–	–	Separated	3	0
83. Multicopy *CDC15*	Viable	–	–	–	–	–
84. *cdc15-ts* multicopy *TEM1*	Inviable	–	–	–	–	–
85. *cdc15*Δ *net1-ts*	Viable	–	–	–	–	–
86. *cdc15-ts* multicopy *CDC14*	Viable	–	–	–	–	–
87. *net1-ts*	Viable	50	–	–	–	–
88. *GAL-NET1*	Inviable	–	–	Separated	3	0
89. *cdc14-ts*	Inviable	–	–	Separated	3	0
90. *GAL-CDC14*	Inviable	–	–	Licensed	5	0
91. *GAL-NET1 GAL-CDC14*	Viable	–	–	–	–	–
92. *net1*Δ *cdc20-ts*	Inviable	–	–	Aligned	1	–
93. *cdc14-ts GAL-SIC1*	Viable	–	–	–	–	–
94. *TAB6-1*	Viable	–	–	–	–	–
95. *TAB6-1 cdc15*Δ	Viable	–	–	–	–	–
96. *TAB6-1 clb5*Δ *clb6*Δ	Inviable	–	–	Licensed	5	0
97. *TAB6-1 CLB1 clb2*Δ	Viable	–	–	–	–	–
98. *mad2*Δ	Viable	35	1	–	–	–
99. *bub2*Δ	Viable	35	1	–	–	–
100. *mad2*Δ *bub2*Δ	Viable	–	–	–	–	–
101. *APC-A*	Viable	20	1.5	–	–	–
102. *APC-A cdh1*Δ	Inviable	–	–	Separated	3	–
103. *APC-A cdh1*Δ in galactose	Viable	–	–	–	–	–
104. *APC-A cdh1*Δ multicopy *SIC1*	Viable	–	–	–	–	–
105. *APC-A cdh1*Δ *GAL-SIC1*	Viable	–	–	–	–	–
106. *APC-A cdh1*Δ multicopy *CDC6*	Viable	–	–	–	–	–
107. *APC-A cdh1*Δ *GAL-CDC6*	Viable	–	–	–	–	–
108. *APC-A cdh1*Δ multicopy *CDC20*	Viable	–	–	–	–	–
109. *swi5*Δ	Viable	20	–	–	–	–
110. *sic1*Δ *cdc6*Δ*2-49 cdh1*Δ	Inviable	–	–	Fired	3	1
111. *sic1*Δ *cdc6*Δ*2-49 cdh1*Δ *GALL-CDC20*	Viable	–	–	–	–	–
112. *APC-A cdh1*Δ *clb5*Δ	Inviable	–	–	–	–	–
113. *APC-A cdh1*Δ *pds1*Δ	Inviable	–	–	–	–	–
114. *APC-A sic1*Δ	Viable	–	–	–	–	–
115. *APC-A GAL-CLB2*	Inviable	–	–	Separated	3	–

References

1. Allen, N.A., Calzone, L., Chen, K.C., Ciliberto, A., Ramakrishnan, N., Shaffer, C.A., Sible, J.C., Tyson, J.J., Vass, M.T., Watson, L.T., Zwolak, J.W.: Modeling regulatory networks at Virginia Tech. OMICS **7**, 285–299 (2003)
2. Allen, N.A., Chen, K.C., Tyson, J.J., Shaffer, C.A., Watson, L.T.: Computer evaluation of network dynamics models with application to cell cycle control in budding yeast. IEE Syst. Biol. **153**, 13–21 (2006)

3. Allen, N.A., Shaffer, C.A., Ramakrishnan, N., Vass, M.T., Watson, L.T.: Improving the development process for eukaryotic cell cycle models with a modeling support environment. Simulation **79**, 674–688 (2003)
4. Audet, C., Dennis, Jr., J.E.: Mesh adaptive direct search algorithms for constrained optimization. SIAM J. Optim. **17**, 188–217 (2006)
5. Chen, K.C., Calzone, L., Csikasz-Nagy, A., Cross, F.R., Novak, B., Tyson, J.J.: Integrative analysis of cell cycle control in budding yeast. Mol. Biol. Cell **15**, 3841–3862 (2004)
6. Coope, I.D., Price, C.J.: Frame based methods for unconstrained optimization. J. Optim. Theory Appl **107**, 261–274 (2000)
7. He, J., Sosonkina, M., Shaffer, C.A., Tyson, J.J., Watson, L.T., Zwolak, J.W. : A hierarchical parallel scheme for a global search algorithm. In: Meyer, J. (ed.) Proceedings of High Performance Computing Symposium 2004, pp. 43–50. Society for Modeling and Simulation International, San Diego, CA (2004)
8. He, J., Sosonkina, M., Shaffer, C.A., Tyson, J.J., Watson, L.T., Zwolak, J.W.: A hierarchical parallel scheme for global parameter estimation in systems biology. In: Proceedings of the 18th International Parallel & Distributed Processing Symposium, 9 p, CD-ROM, Los Alamitos, CA, IEEE Computer Soc. (2004)
9. He, J., Sosonkina, M., Watson, L.T., Verstak, A., Zwolak, J.W.: Data-distributed parallelism with dynamic task allocation for a global search algorithm. In: Parashar, M., Watson, L. (eds.) Proceedings of High Performance Computing Symposium 2005, pp. 164–172. Society for Modeling and Simulation International, San Diego, CA (2005)
10. He, J., Watson, L.T., Ramakrishnan, N., Shaffer, C.A., Verstak, A., Jiang, J., Bae, K., Tranter, W.H.: Dynamic data structures for a direct search algorithm. Comput. Optim. Appl. **23**, 5–25 (2002)
11. Jones, D.R., Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the Lipschitz constant. J. Optim. Theory Appl. **79**, 157–181 (1993)
12. Murray, A., Hunt, T.: The Cell Cycle: an Introduction. Oxford University Press, New York (1993)
13. Nasmyth, K.: At the heart of the budding yeast cell cycle. Trends Genet. **12**, 405–412 (1996)
14. Nurse, P.: A long twentieth century of the cell cycle and beyond. Cell **100**, 71–78 (2000)