dFBA Formulation

The reactions described within an organism's GSMM are organized into a stoichiometric matrix, *S*. The amount of material that moves through each reaction is known as metabolic "flux", represented by v. Each reaction is represented by an equality constraint that assumes steady state: $Sv = 0$. Further inequality constraints are applied to represent nutrient concentrations. The full linear program (LP) formulation is as follows, with lower and upper bounds on flux, converted into standard form for solution by simplex method[51]:

$$\begin{aligned} &\mu = \max \boldsymbol{c}^T \boldsymbol{v} \\ &st. \boldsymbol{S}\boldsymbol{v} = 0 \\ &\quad \boldsymbol{v}_L \le \boldsymbol{v} \le \boldsymbol{v}_U \end{aligned} \longrightarrow \begin{aligned} &\mu = \min \bar{\boldsymbol{c}}^T \bar{\boldsymbol{v}} \\ &st.\ \boldsymbol{S}\bar{\boldsymbol{v}} = \boldsymbol{b} \\ &\quad \bar{\boldsymbol{v}} \ge 0 \end{aligned} \qquad (1)$$

As mentioned in **Section 3.**2, conventional dFBA solvers repeatedly solve an LP at every point in the initial value problem. Simulators developed by Höffner[52], Gomez[53], et al. convert the ODE/LP system into a system of differential algebraic equations (DAEs). These simulators save an LP solution, and at every time step, scale the solution based on the new constraints. The LP solution only needs to be calculated if the LP solution is infeasible given the current time step's constraints, because an optimal solution remains optimal until one of its inactive constraints is violated.[54]

The dFBA solvers mentioned above are written in FORTRAN and MATLAB, respectively; in addition, the solvers are geared toward batch and fed-batch simulation. Due to these specifications, and the fact that our solver needs to change objective functions frequently (day/night cycle), we have chosen to adapt the methods used by Höffner, Gomez, et al. for our needs instead of using their method out-of-the-box. Specifically, since our model is focused almost entirely on the process of photosynthesis, there is frequently a single limiting factor. For most of the simulations the limiting factor is carbon uptake via RuBisCo. Therefore, the saved

solution can simply be scaled by the saved and current CO_2 bound. When other micronutrients become limiting, our implementation detects that and changes the limiting nutrient. When NO_3 or $Si(OH)_4$ or iron are limiting, the same process is used: scale by the previous and current bound. The only complication to this procedure is when carbon is the limiting step for the symbiotic pair. Since carbon is transferred from the host diatom to the cyanobacterial symbiont, solution scaling is not as straightforward. The scaling factor for each model component is comprised by one of three model constraints: diatom CO_2, cyanobacterial CO_2, and the carbon transfer constraint, as shown below, where D_CO_2 is the current/saved diatom CO_2 bound, C_CO_2 is the cyanobacterial CO_2 bound, and CT is the carbon transfer constraint:

Variable	**Scaling Factor (bounds)**
Diatom Biomass	D_CO_2 + C_CO_2
Cyano Biomass	D_CO_2 + C_CO_2
Cyano CO_2	C_CO_2
Diatom CO_2	D_CO_2
Diatom O_2	D_CO_2
Cyano O_2	CT + C_CO_2
Diatom $Si(OH)_4$	D_CO_2 + C_CO_2
Cyano Fe	D_CO_2 + C_CO_2
Cyano N_2	D_CO_2 + C_CO_2

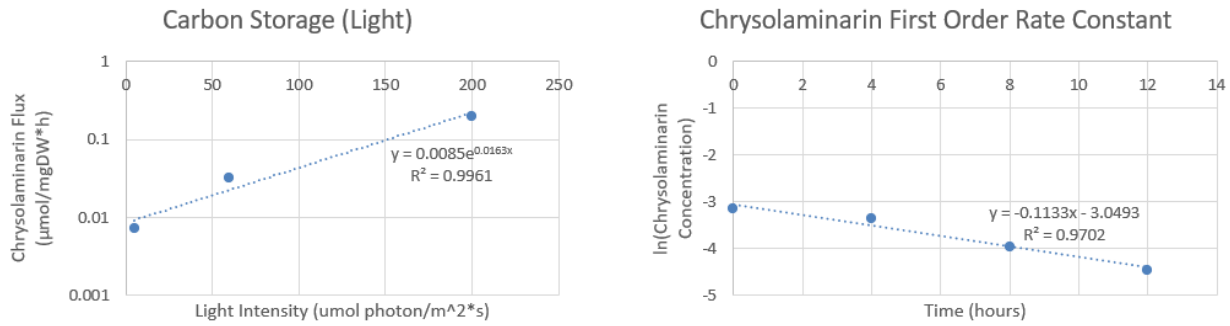
In addition to the computational efficiency improvement afforded by avoiding LP calculation, Höffner, Gomez, et al. solve one additional problem with their solvers: non-uniqueness of solutions. In linear programming, the optimal cost is guaranteed to be unique, but the other decision variables' values are not.[55] The result is a continuous but nonsmooth mapping

of the right-hand side of the ODEs. Implicit differential equation integration methods have assumptions that require a smooth righthand side.[56] Implicit algorithms are the first line of defense against stiff systems of ODEs, so naïve LP solution is very restrictive, and can lead to large increases in computational time. To solve this problem, Höffner et al. propose a hierarchical optimization in which any dynamic nutrients (included in ODE) are optimized sequentially, while holding all previous optimal values equal:

$$
\begin{aligned}
g_2 &= \min \bar{\boldsymbol{c}}_2{}^T \bar{\boldsymbol{v}} \\
&s.t. \boldsymbol{A}\bar{\boldsymbol{v}} = \boldsymbol{b},\ \mu_1 = \mu_2,\ \bar{\boldsymbol{v}} \geq 0 \\
g_3 &= \min \bar{\boldsymbol{c}}_3{}^T \bar{\boldsymbol{v}} \\
&s.t. \boldsymbol{A}\bar{\boldsymbol{v}} = \boldsymbol{b},\ \mu_1 = \mu_2, v_{g_2} = v_{g_3},\ \bar{\boldsymbol{v}} \geq 0
\end{aligned}
\tag{2}
$$

The dynamic variables are different during the day and night (CO_2 and O_2 are day-only, while chrysolaminarin is night-only), requiring changes in the objective functions within the hierarchical optimization. See the code within our supplemental code file for the implementation.

Chrysolaminarin Calibration Curves



The daytime calibration curve (left) is calculated from data from Fisher and Halsey, processed by van Tol et al. in their genome scale metabolic model. By plugging in the total light intensity from the light intensity calculator, we can find the accumulation rate of chrysolaminarin during the daytime. The nighttime calibration curve (right) is calculated from data from Liu et al.

Chrysolaminarin consumption appears to follow first-order kinetics at night, so we find the rate constant and use it within our model.