Theory Paper: Supplementary Information

Grant Kinsler, Kerry Geiler-Samerotte, Dmitri Petrov

1 Mathematical and Computational Details

1.1 Derivation of Score Metric

To derive the score (eq. 1) used in the main text, we assume that fitness for mutant j at position $\vec{x}_j$ in condition k is given by:

$$F_{jk} = \frac{h_k}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{\sum_{i=1}^{D}\left(x_{ij}-o_{ik}\right)^2}{2\sigma_k^2}\right) \tag{1}$$

where fitness in condition k is represented by a Gaussian function centered at $\vec{o}_k$, with height h_k and variance σ_k^2. Using this absolute fitness function, relative fitness from an ancestor at the location $\vec{a}$ is given by:

$$f_{jk} = \frac{F_{jk}}{F_{Ak}} - 1 = \frac{\exp\left(-\frac{\sum_{i=1}^{D}(x_{ij}-o_{ik})^2}{2\sigma_k^2}\right)}{\exp\left(-\frac{\sum_{i=1}^{D}(a_i-o_{ik})^2}{2\sigma_k^2}\right)} - 1 = \exp\left(\frac{\sum_{i=1}^{D}\left(a_i-o_{ik}\right)^2 - \sum_{i=1}^{D}\left(x_{ij}-o_{ik}\right)^2}{2\sigma_k^2}\right) - 1 \tag{2}$$

Note that the height of the Gaussian h_k drops out of the equation for relative fitness. To find values of the parameters (including location of the mutants, optima, and ancestor) that best fit our observed data (the measured relative fitnesses), we need to derive a score that minimizes the difference between the predicted relative fitness given by the model and the observed relative fitness. Since our estimation technique relies on repeatedly evaluating a function of the locations of the mutants, optima, and ancestor, it is computationally costly to have an exponential function in this evaluation. Accordingly, we use a log-transform of this equation to derive our score metric:

$$S\left(x,o,a|f,\epsilon\right) = \sum_{j=1}^{M}\sum_{k=1}^{C}\epsilon_{jk}\left[\left(\sum_{i=1}^{D}\left(a_i-o_{ik}\right)^2 - \sum_{i=1}^{D}\left(x_{ij}-o_{ik}\right)^2\right) - 2\sigma_k^2\log\left[1+f_{jk}\right]\right]^2 \tag{3}$$

where the ϵ_{jk} is weighting for measurement error according to relative inverse variance weighting, such that:

$$\epsilon_{jk} = \frac{\frac{1}{s_{jk}^2}}{\sum_{j,k}\frac{1}{s_{jk}^2}}$$

where s_{jk}^2 is the sampling variance for the given measurement [CITE venkataram supplement?].

1.2 Constraining symmetry

There are a number of symmetric solutions that can fit this solution. We constrain symmetry and reduce the number of parameters that we fit by the ancestor to the location $(a, 0, \ldots, 0)$ and optima such that the first optimum is at the origin, and the dth optimum is constrained to the

d-dimensional Euclidean subspace (up until the Dth optimum, whereafter each optimum has all D values). After imposing these constraints, and fixing the variance of the Gaussian to $\sigma_k^2 = 1.0$, we can calculate the total number of variables in our optimization problem to understand when the problem becomes underdetermined. We need the number of variables to be less than the number of data points (which is given by MC). Thus we are overdetermined (and okay solving the problem if):

$$M(D) + (C - D)D + \frac{D(D+1)}{2} < MC$$

When there are many more mutants than conditions, then the terms with M dominate, and the problem is feasible when the number of dimensions is less than the number of conditions included.

1.3 Optimization Implementation

To find the best possible solution to this nonlinear optimization problem, we use a global minimization technique implemented in Python. In particular, we use `scipy.optimize.basinhopping`, which combines gradient descent with random displacement to find various local minima. Local minima are accepted similarly to Metropolis-Hastings and Simulated Annealing approaches, and the process continues to iterate until the number of steps has been completed. We run numerous simulations from random starting locations and compare the found solutions to ensure consistency in our optima. However, because this is a global optimization technique, there is no guarantee that the solution we find is the best possible one. Thus, our solutions should be seen as the best local optimum found. Because of this, substantial simulations are needed to confirm the robustness of this model and its implementation.

1.4 Simulation Implementation

To generate simulation data, we uniformly draw points from within the n-dimensional ball according to the Marsaglia Algorithm [**?**]