SOFTWARE

JuPOETs: A Constrained Multiobjective Optimization Approach to Estimate Biochemical Model Ensembles in the Julia Programming Language

David Bassen[2], Jonathan T Butcher[2] and Jeffrey D Varner[1*]

[*]Correspondence:
jdv27@cornell.edu
[1]Department of Chemical and Biomolecular Engineering, Cornell University, 14806 Ithaca NY, USA
Full list of author information is available at the end of the article

Abstract

First part title: Text for this section.

Second part title: Text for this section.

Keywords: sample; article; author

Background

Models of signal transduction networks may exhibit complex relationships between model performance and parameter values [1]. It is often not possible to uniquely identify parameters from noisy experimental measurements, even when given extensive training data [2]. Thus, despite significant advances in standardizing model identification [3], the core problem of estimating model parameters from noisy data remains. Ensemble approaches have been used to address parameter uncertainty in systems biology and other fields like weather prediction [4, 5, 6, 7]. In an ensemble approach, a *family* of approximate parameter sets is identified instead of a single best-fit parameter set. Parameter families can be selected based on simulation error, along with other criterion such as diversity. Simulations using parameter ensembles can estimate confidence intervals on model variables, and robustly constrain model predictions, despite having many poorly constrained parameters [8, 9]. There are several techniques to generate parameter ensembles. Battogtokh et al., Brown et al., and later Tasseff et al. generated experimentally constrained parameter ensembles using a Metropolis-type random walk through parameter space [4, 6, 10, 11]. Other strategies could also be adapted to generate parameter ensembles [12]. However, the unifying component of all of these previous strategies was the minimization of a single objective function.

Identification of large models with hundreds or even thousands of states and parameters may not be tractable as a single objective formulation. Models at these scales require significant training data perhaps taken from diverse sources, for example different laboratories or cell-lines. These diverse training data are likely heterogenous, and contain intrinsic conflicts. On the other hand, large models can also be constructed from many smaller modules. For example, Diamond and coworkers developed a model of phosphoinositide and calcium signaling in human platelets composed of many separate submodels identified using different types of training

data [13]. Taken together, the problem of identifying parameter ensembles for large-scale models with many parameters posses a unique challenge. Parameter ensemble estimation techniques which optimally balance tradeoffs between submodels or conflicts in training data might lead to more robust model performance. One class of such techniques is multiobjective optimization. Previously, we developed the Pareto Optimal Ensemble Technique (POETs) algorithm to address the challenge of competing or conflicting objectives. POETs, which integrates simulated annealing and multiobjective optimization through the notion of Pareto rank, finds a family of parameter estimates which optimally trade-off between competing (and potentially conflicting) experimental objectives [14]. POETs has been used in several biochemical and signal transduction studies to estimate parameter ensembles [15, 16]. However, the previous implementation of POETs, in the Octave programming language [17], suffered from poor performance, and did not offer user definable functions such as custom cooling schedules, parameter constraints, or custom search logic. It was also not well integrated into a modern package or source code management (SCM) system.

Implementation

In this study, we present JuPOETs, an open-source implementation of the Pareto Optimal Ensemble Technique (POETs) in the Julia programming language. JuPOETs offers many advantages and improvements compared to the previous implementation of POETs. First, JuPOETs takes advantage of the unique performance features of the Julia programming language. Julia, which has performance comparable to C but with syntax similar to MATLAB/Octave and Python, is a cross-platform, high-performance programming language for technical computing [18]. Julia offers a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library. Additionally, Julia offers a built-in package manager which is directly integrated with GitHub, a popular web-based Git repository hosting service which offers distributed revision control and source code management. Next, because Julia can natively call other languages such as Python or C, JuPOETs can be used with models implemented in a variety of languages on many platforms. Lastly, the architecture of JuPOETs takes advantage of the first-class function type in Julia allowing user definable behavior for all key aspects of the algorithm, including objective functions, custom search logic, linear/nonlinear parameter constraints (and parameter bounds constraints) as well as custom cooling schedule functions. Thus, JuPOETs can easily be adapted to solve many problem types, including mixed binary and continuous variable types, without the need to change the base algorithm (which was not true of the previous POETs implementation).

Optimization problem formulation.

JuPOETs solves the $\mathcal{K}-$dimensional constrained multiobjective optimization problem:

$$\min_{\mathbf{p}} \begin{cases} O_1\left(\mathbf{x}(t,\mathbf{p}),\mathbf{p}\right) \\ \vdots \\ O_{\mathcal{K}}\left(\mathbf{x}(t,\mathbf{p}),\mathbf{p}\right) \end{cases} \tag{1}$$

subject to:

$$\begin{aligned} \mathbf{f}(t, \mathbf{x}(t,\mathbf{p}), \dot{\mathbf{x}}(t,\mathbf{p}), \mathbf{u}(t), \mathbf{p}) &= \mathbf{0} \\ g_1\left(t, \mathbf{x}(t,\mathbf{p}), \mathbf{u}(t), \mathbf{p}\right) &\geq 0 \\ &\vdots \\ g_{\mathcal{C}}\left(t, \mathbf{x}(t,\mathbf{p}), \mathbf{u}(t), \mathbf{p}\right) &\geq 0 \end{aligned}$$

and parameter bound constraints:

$$\mathcal{L} \leq \mathbf{p} \leq \mathcal{U}$$

The term t denotes time, $g_i\left(t, \mathbf{x}(t,\mathbf{p}), \mathbf{u}(t), \mathbf{p}\right)$ denotes the ith constraint on the search, where $\mathbf{x}\left(t, \mathbf{p}\right)$ denotes the model state variable vector with an initial state $\mathbf{x}_0$, and $\mathbf{u}(t)$ is an input vector. The terms $\mathbf{f}(t, \mathbf{x}(t,\mathbf{p}), \mathbf{u}(t), \mathbf{p})$ denote the system of model equations (e.g., differential equations, differential algebraic equations or algebraic constraints) and $\mathbf{p}$ denotes the unknown parameter vector ($\mathcal{P} \times\ 1$). The parameter search can be subject to parameter bound constraints where $\mathcal{L}$ and $\mathcal{U}$ denote the lower and upper parameter bounds, respectively.

POETs integrate Simulated Annealing (SA) with Pareto optimality to estimate parameter sets on or near the optimal tradeoff surface between competing training objectives (Fig. 1). The central idea underlying POETs is a mapping between the value of the objective functions evaluated at $\mathbf{p}_{i+1}$ (parameter guess at iteration $i+1$) and the Pareto rank. We computed the Pareto rank of $\mathbf{p}_{i+1}$ by comparing the simulation error at iteration $i+1$ against an archive of simulation errors compiled up to iteration i $\mathbf{E}_i$. We used the Fonseca and Fleming scheme to compute the Pareto rank, r [19]:

$$rank\left(\mathbf{p}_{i+1} \mid \mathbf{E}_i\right) = r \tag{2}$$

where r denotes the rank, or the number of parameter sets that dominate parameter set $\mathbf{p}_{i+1}$. Parameter sets on the optimal trade-off surface between the objectives have a rank equal to 0 (no other current parameter sets are better). Sets with increasing non-zero rank are progressively further away from the optimal trade-off surface. Parameter sets on the optimal trade-off surface have a rank equal to 0 (no other current parameter sets are better). Sets with increasing non-zero rank are progressively farther away from the optimal trade-off surface. Thus, a parameter set with a $rank = 0$ is *better* in a trade-off sense than $rank > 0$. We used the Pareto rank to inform the SA calculation. The parameter set $\mathbf{k}_{i+1}$ was accepted or rejected by the SA using the acceptance probability $\mathcal{P}\left(\mathbf{k}_{i+1}\right)$:

$$\mathcal{P}(\mathbf{k}_{i+1}) \equiv \exp\left\{-rank\left(\mathbf{k}_{i+1} \mid \mathbf{K}_i\right)/T\right\} \tag{3}$$

where T is the computational annealing temperature. As $rank\left(\mathbf{k}_{i+1} \mid \mathbf{K}_i\right) \rightarrow 0$, the acceptance probability moved toward one, ensuring that we explored parameter sets along the Pareto surface. Occasionally (depending upon T) a parameter set

with a high Pareto rank was accepted by the SA allowing a more diverse search of the parameter space. However, as T was reduced, the probability of this occurring decreased. Parameter sets could be accepted by the SA and *not* archived in $\mathbf{K}_i$. Only parameter sets with rank less than or equal to a user defined threshold (rank$\leq$4 by default) were included in $\mathbf{K}_i$ to ensure that we characterized the neighborhood near the trade-off surface (Algorithm 1).

input : User specified neighbor, objective, acceptance and cooling functions. Initial parameter guess ($\mathcal{P} \times 1$)
Output: Rank archive $\mathcal{R}$, solution archive $\mathcal{S}$ and objective archive $\mathcal{O}$

initialize: $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{O}$ using initial guess;
initialize: T $\leftarrow$1.0;
initialize: T_{min} $\leftarrow$1/10000;
initialize: Maximum number of steps per temperature $\mathcal{I}$;

while $T > T_{min}$ **do**
    i $\leftarrow$ 1;
    while $i < \mathcal{I}$ **do**
        `// Generate a new parameter solution using user neighbor function`
        $\mathbf{p}_{i+1} \leftarrow$ user-function::`neighbor`($\mathbf{p}^*$);
        `// Evaluate` $\mathbf{p}_{i+1}$ `using user objective function`
        $\mathbf{o}_{i+1} \leftarrow$ user-function::`objective`($\mathbf{p}_{i+1}$);
        Add $\mathbf{p}_{i+1}$ to solution archive $\mathcal{S}$;
        Add $\mathbf{o}_{i+1}$ to objective archive $\mathcal{O}$;
        `// Calculate Pareto rank of solutions in` $\mathcal{O}$ `using builtin rank function`
        $\mathcal{R} \leftarrow$ builtin-function::`rank`($\mathcal{O}$);
        `// Accept` $\mathbf{p}_{i+1}$ `into the archive with user defined probability`
        $\mathcal{P} \leftarrow$ user-function::`acceptance`($\mathcal{R}$,T);
        if $\mathcal{P} > rand$ **then**
            `// Update the best solution with` $\mathbf{p}_{i+1}$
            $\mathbf{p}^* \leftarrow \mathbf{p}_{i+1}$;
            prune $\mathcal{S}$, $\mathcal{R}$ and $\mathcal{O}$ of all solutions above a rank threshold;
        else
            Remove $\mathbf{p}_{i+1}$ from solution archive $\mathcal{S}$;
            Remove $\mathbf{o}_{i+1}$ from error archive $\mathcal{O}$;
        end
        i $\leftarrow$ i + 1;
    end
    `// Update` T `using the user cooling function`
    $T \leftarrow$ user-function::cooling(T);
end

Algorithm 1: Pseudo-code for the run-loop of the JuPOETs algorithm. The user specifies the neighbor, acceptance, cooling and objective function pointers along with a initial parameter guess. The rank archive $\mathcal{R}$, solution archive $\mathcal{S}$ and objective archive $\mathcal{O}$ are initialized from the initial guess. The initial guess is then perturbed in the neighbor function, which generates a new solution whose performance is evaluated using the user supplied objective function. The new solution and objective values are then added to the respective archives and ranked using the builtin rank function. If the new solution accepted (based upon a probability calculated with the user supplied acceptance function) it is added to the solution and objective archive. This solution is then perturbed during the next iteration of the algorithm. However, if the solution is not accepted, it is removed from the archive and discarded. The computational temperature is adjusted using the user supplied cooling function after each $\mathcal{I}$ iterations.

Results and Discussion

JuPOETs identified solutions on or near the optimal tradeoff surface significantly faster than Octave-POETs for the suite of test problems (Fig. 2 and Fig. 3). We compared the performance of JuPOETs versus POETs implemented in Octave for a suite of multiobjective test functions (Table 1). The execution time (wall-clock) for JuPOETs and Octave-POETs was measured for 10 independent trials for the suite of test problems. The same cooling, neighbor, acceptance, search and objective logic was employed between the implementations, with all other parameters held constant. For each test function, the search domain was partitioned into 10 segments, where initial parameter guesses were drawn from each partition. The number of search steps for each temperate $\mathcal{I} = 10$ for all cases, and the cooling parameter was $\alpha = 0.9$. On average, JuPOETs identified optimal or near optimal solutions for the suite of test problems six-fold faster (60s versus 400s) than Octave-POETs (Fig. 2). In addition, JuPOETs produced the characteristic tradeoff curves for each test problem, given both parameter bound and problem constraints (Fig. 3). Taken together, these results suggested that JuPOETs was able to estimate an ensemble of solutions to constrained multiobjective optimization problems significantly faster than the current Octave implementation. However, we have not demonstrated JuPOETs on a biochemically relevant test problem.

[JuPOETs and a proof-of-principle HCM-EM model goes here]

Conclusions

Competing interests
The authors declare that they have no competing interests.

Funding
This study was supported by an award from the National Science Foundation (NSF CBET-0955172) and the National Institutes of Health (NIH HL110328) to J.B, and by a National Science Foundation Graduate Research Fellowship (NSGRF) to D.B.

Author's contributions
J.V developed the software presented in this study. The manuscript was prepared and edited for publication by D.B, J.B. and J.V.

Author details
[1]Department of Chemical and Biomolecular Engineering, Cornell University, 14806 Ithaca NY, USA. [2]Department of Biomedical Engineering, Cornell University, 14806 Ithaca NY, USA.

References

1. K. S. Brown, G.A.C.C.R.M.K.H.L.J.P.S.R.A.C. C. C. Hill: The statistical mechanics of complex signaling networks: nerve groth factor signaling. Phys Biol **1**, 184–195 (2004)
2. Gadkar, K.G., Varner, J., Doyle, F.J.: Model identification of signal transduction networks from data using a state regulator problem. Syst Biol (Stevenage) **2**(1), 17–30 (2005)
3. Gennemark, P., Wedelin, D.: Benchmarks for identification of ordinary differential equations from time series data. Bioinformatics **25**(6), 780–6 (2009). doi:10.1093/bioinformatics/btp050
4. D Battogtokh, M.E.C.J.A.H.-B.s. D K Asch: An ensemble method for identifying regulatory circuits with special reference to the qa gene cluster of neurospora crassa. P Natl Acad Sci USA **99**(26), 16904–16909 (2002)
5. L. Kuepfer, U.S.J.S. M. Peter: Ensemble modeling for analysis of cell signaling dynamics. Nat Biotech **25**(9), 1001–1006 (2007)
6. K S Brown, J.P.S.: Statistical mechanial approaches to models with many poorly known parameters. Phys Rev E **68**, 021904–19 (2003)
7. T. N. Palmer, R.H.F.J.D.-R.T.J.M.L. G. J. Shutts: Representing model uncertainty in weather and climate prediction. Ann Rev Earth and Planetary Sci **33**, 163–193 (2005)
8. Gutenkunst, R.N., Waterfall, J.J., Casey, F.P., Brown, K.S., Myers, C.R., Sethna, J.P.: Universally sloppy parameter sensitivities in systems biology models. PLoS Comput Biol **3**(10), 1871–1878 (2007). doi:10.1371/journal.pcbi.0030189
9. Song, S., Varner, J.: Modeling and Analysis of the Molecular Basis of Pain in Sensory Neurons. PLoS ONE **4**, 6758–6772 (2009)

10. Tasseff, R., Nayak, S., Salim, S., Kaushik, P., Rizvi, N., Varner, J.D.: Analysis of the molecular networks in androgen dependent and independent prostate cancer revealed fragile and robust subsystems. PLoS One **5**(1), 8864 (2010). doi:10.1371/journal.pone.0008864
11. Tasseff, R., Nayak, S., Song, S.O., Yen, A., Varner, J.D.: Modeling and analysis of retinoic acid induced differentiation of uncommitted precursor cells. Integr Biol (Camb) **3**(5), 578–91 (2011). doi:10.1039/c0ib00141d
12. C. G. Moles, J.R.B. P. Mendes: Parameter estimation in biochemical pathways: a comparison of global optimization methods. Genome Res **13**, 2467–2474 (2003)
13. J. E. Purvis, L.F.B.S.L.D. M. S. Chatterjee: A molecular signaling model of platelet phosphoinositide and calcium regulation during homeostasis and p2y 1 activation. Blood **112**(10), 4069–4079 (2008)
14. Song, S.O., Chakrabarti, A., Varner, J.D.: Ensembles of signal transduction models using pareto optimal ensemble techniques (poets). Biotechnol J **5**(7), 768–80 (2010). doi:10.1002/biot.201000059
15. Song, S.O., Varner, J.: Modeling and analysis of the molecular basis of pain in sensory neurons. PLoS One **4**(9), 6758 (2009). doi:10.1371/journal.pone.0006758
16. Lequieu, J., Chakrabarti, A., Nayak, S., Varner, J.D.: Computational modeling and analysis of insulin induced eukaryotic translation initiation. PLoS Comput Biol **7**(11), 1002263 (2011). doi:10.1371/journal.pcbi.1002263
17. https://www.gnu.org/software/octave/ Accessed May 11, 2016
18. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing (2015). 1411.1607
19. Fonseca, C.M., Fleming, P.J.: Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In: Proceedings of the 5th International Conference on Genetic Algorithms, pp. 416–423 (1993)

Figure 1: Schematic of multiobjective parameter mapping. The performance of any given parameter set is mapped into an objective space using a ranking function which quantifies the quality of the parameters. The distance away from the optimal tradeoff surface is quantified using the Pareto ranking scheme of Fonseca and Fleming in JuPOETs.

Figure 2: The performance of JuPOETs on the multi-objective test suite. The execution time (wall-clock) for JuPOETs and POETs implemented in Octave was measured for 10 independent trials for the suite of test problems. The number of steps per temperature $\mathcal{I} = 10$, and the cooling parameter $\alpha = 0.9$ for all cases. The problem domain was partitioned into 10 equal segments, an initial guess was drawn from each segment. For each of the test functions, JuPOETs estimated solutions on (rank zero solutions, black) or near (gray) the optimal tradeoff surface, subject to bounds and problem constraints.

Figures

Figure 3: Representative JuPOETs solutions for problems in the multi-objective test suite. The number of steps per temperature $\mathcal{I} = 10$, and the cooling parameter $\alpha = 0.9$ for all cases. The problem domain was partitioned into 10 equal segments, an initial guess was drawn from each segment. For each of the test functions, JuPOETs estimated solutions on (rank zero solutions, black) or near (gray) the optimal tradeoff surface, subject to bounds and problem constraints.

Table 1: Multi-objective optimization test problems. We tested the JuPOETs implementation on three two-dimensional test problems, with one-, two- and three-dimensional parameter vectors. Each problem had parameter bounds constraints, however, on the Binh and Korn function had additional non-linear problem constraints. For the Fonesca and Fleming problem, N = 3.

Tables