A Combinatorial Design Workflow for Search and Prioritization in Large-Scale Synthetic Biology Construct Assembly

Justin Ng, Aaron Berliner, Joe Lachoff, Florencio Mazzoldi, Eli Groban
Autodesk Research
jkc2ng@uwaterloo.ca, {aaron.berliner, joe.lachoff, florencio.mazzoldi, eli.groban}@autodesk.com

Abstract

We propose a strategy to effectively search the large solution space for a genetic system built from combinations of genetic parts. An iterative and interchangeable algorithm then searches the combinatorial space and creates a reasonably sized test set to build using new highly parallel genetic foundry capabilities. The algorithm can also learn from information gained from prior iterations to suggest combinations to try in the following iteration.

1 Introduction

The commercialization of large scale DNA construction companies such as Amyris and Intrexon and their academic counterparts such as The Foundry at Imperial College London and the Edinburgh Genome Foundry enable new approaches to synthetic biology design. They grant synthetic biologists the ability to easily build and test large genetic constructs in excess of 30,000 bp, while abstracting away the construction techniques. Furthermore, these foundries are able to assemble thousands of these constructs in parallel. However, utilizing these capabilities adds additional complexity when designing large quantities of genetic constructs of this size. Software tools such as the Genotype Specification Language (GSL) have been designed to help biologists leverage the new capabilities of these foundries, while keeping complexity to a minimum. GSL provides a formal language for a higher level of genetic specification and aims to move the focus from construction to function [Wilson *et al.*, 0]. Autodesk is producing a cloud based, extensible platform for genetic design that aims to reduce the complexity when designing large sequences. However, both of these tools aim to help the biologist by managing the complexity of longer sequences, and few tools have been targeted at managing the complexity of creating a large number of sequences. In this paper, we describe a workflow to aid biologists that enables the semi-rational design of sequences that can leverage the parallel nature of foundries. This workflow provides a method for the optimization of a performance metric for a combination of genetic parts.

2 Combinatorial Design

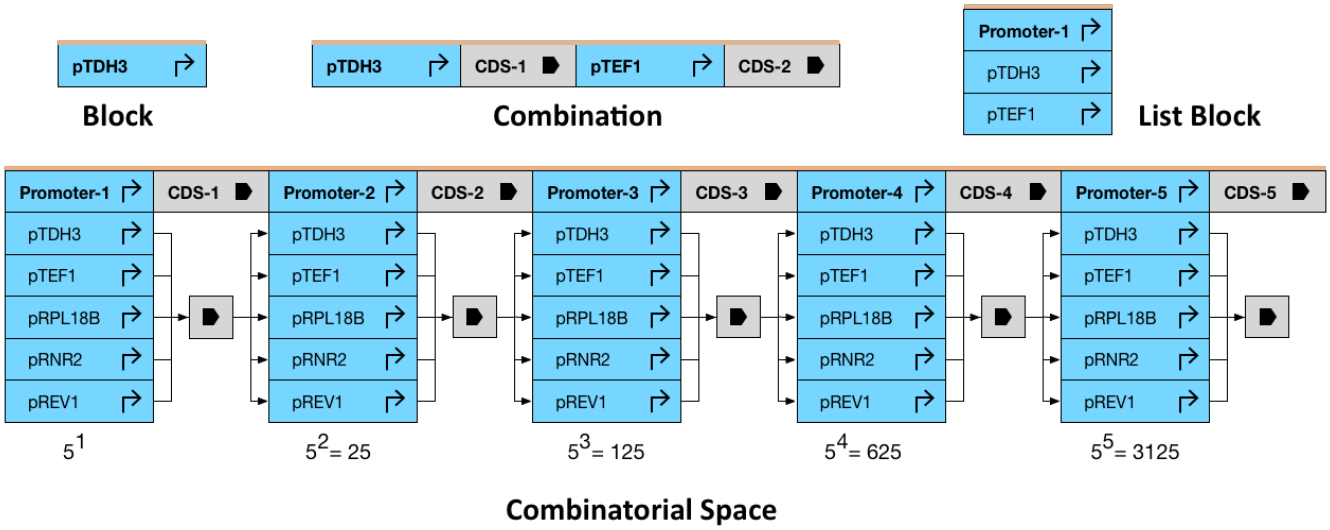
Optimization of genetic systems is currently done either through screening experiments or done manually. Screening experiments are undirected and require an exponential increase in combinations tried to gain an expected improvement in performance, while manual experimentation requires a considerable amount of cognitive effort, an understanding of the system, and most often fail to produce the desired result. We present the alternative of a semi-rational design.

2.1 Design Specification

Combinatorial designs have been used in screening experiments to efficiently optimize the performance of genetic systems [Smanski *et al.*, 2014]. However, these methods often require in depth knowledge of construction techniques and screening experiments are not always possible due to biological conflict or inherent limitations on the screening numbers. Only a finite number of combinations can be screened at a given time and this number is limited by the laboratory automation configuration. Our approach in dealing with this limitation is to prioritize the designs for testing. We describe a semi-rational process, where a user specifies a given combinatorial design, from which an algorithm chooses prioritized combinations of parts to assemble and test.

For our definition of combinatorial design, we first build a concept of *blocks*, and extend them to *list blocks*. A block represents a DNA sequence or a genetic part, such as a promoter or a terminator. The complete list of these genetic parts is sourced from the Synthetic Biology Open Language (SBOL) visual framework. Blocks can be composed sequentially with other blocks. List blocks extend this concept by representing multiple blocks, similar in style to a drop-down list. A *combinatorial design* is a sequence of blocks that contains multiple list blocks. A combinatorial design spans a *combinatorial space*, which represents all possible instantiations of the list blocks. These spaces grow extremely quickly in size due to the curse of dimensionality [Keogh and Mueen, 2010], thus synthesizing and testing all possible combinations quickly becomes intractable. As a result of this, a more intelligent search of the space is required. In the rest of this paper, we will use the term combination to refer to a given combination of genetic parts that can be specified by the design. We build our workflow on top of Autodesk Genome Designer,

Figure 1: **Combinatorial Design Space of Promoters for Violacein Pathway.** Figure depicts the graphical difference between Block, Combination, List Block, and Combinatorial Space in terms of their representation as UX elements styled with SBOL Visual symbols. The Combinatorial Space shown depicts the $5^5 = 3125$ combinations of promoters pTDH3, pTEF1, pRPL18B, pRNR2 and pREV1.



which is a drag-and-drop platform for synthetic biology construct design.

3 Combinatorial Design Workflow

We propose a workflow for bioengineers to find the optimal or near-optimal combination of parts. Our workflow mirrors the standard synthetic biology mantra of design/build/test/learn. This workflow, described in Figure 2, is composed of both a software step where the priority of combinations is optimized and an external validation step where combinations are experimentally characterized.

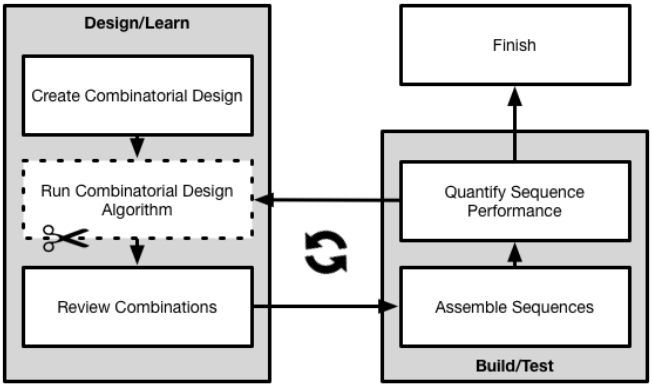
In the software portion, the user creates a design of parts utilizing list blocks, selects the number of combinations they wish to test, and then uses the combinatorial design feature to select the combinations that should be run. The user is given a chance to edit the combinations to their liking, and then the results will be submitted in the form of a bill of materials to a foundry of choice for experimental testing. This software portion corresponds to the design and learn stages of the synthetic biology cycle.

Given a set of combinations, the foundry will synthesize, assemble, and either return them to the user or perform for screening and analysis. Performance of a given combination should return a scalar value used for comparison to other combinations. These results should then be used as input back into the software step, beginning a secondary iteration of optimization. This experimental portion corresponds to the build and test stages of the synthetic biology cycle.

4 Combinatorial Design Programs

The core of this workflow is the algorithm that selects the next batch of combinations for biological evaluation. This

Figure 2: **Proposed Combinatorial Design Workflow.** Figure depicts the workflow for using combinatorial design as a prioritization methodology in the synthetic biology Design-Build-Test-Learn Cycle. This cycle is broken into two segments, Design/Learn and Build/Test, to illustrate the difference between in silico and experimental aspects of iteration. The dotted line illustrates that various algorithms can be cut-and-pasted into the workflow.



algorithm can be seen as an iterative optimization problem over several categorical variables.

Let $\mathcal{X} = \{[\mathcal{D}_1] \times ... \times [\mathcal{D}_d]\}$ be the combinatorial space of possible designs, where $\mathcal{D}_i$ is the number of choices in the ith list block and d is the number of list blocks in the design. Let $\mathcal{D}$ be the set of pairs for all inputs $x \in \mathcal{X}$ and its corresponding real world performance $y \in \mathbb{R}$. We define a dataset $D \subseteq \mathcal{D}$ to be a set of pairs between inputs $x \in \mathcal{X}$ and a performance value $y \in \mathbb{R}$, where the performance values

are known. Let k be the number of combinations to try per iteration.

The algorithm is then a function of the following form:

$$F : D \rightarrow \mathcal{X}^k \tag{1}$$

with the goal:

$$\max_{(x,y)\in\mathcal{D}} y \tag{2}$$

This workflow follows the *Sequential Model-based Global Optimization* (SMBO) algorithm [Bergstra *et al.*, 2011]. SMBO aims to solve $\arg\max_{x\in\mathcal{X}} f(x)$, where $f(x)$ is an expensive, black box function. This is done by iterating between fitting a model and using that model to identify areas with a high probability of improvement. In this instance, $f(x)$ is the real-world performance of a given combination x. The main difference from SMBO and our workflow is that we select multiple points to try, effectively creating a parallel instantiation of SMBO.

We treat each genetic part as a categorical variable for simplicity and generality, as biological data is often non-homogeneous and sparse. Furthermore, modeling such systems has proven difficult, especially beyond the transcriptional level. We currently refrain from using metadata associated with a given part to allow the program to be used with arbitrary parts. This mathematical abstraction also implies that the designs are permutation invariant, so information about the order of blocks is ignored. Furthermore, the abstraction also removes knowledge about the function of the block, such as its SBOL part type. Lastly, we also assume that the performance of a given combination is deterministic.

4.1 Sample Algorithms

Any algorithm that follows has the function type $F : D \rightarrow \mathcal{X}^k$ can be used in this workflow. Programs designed to optimize black box functions which support categorical variables can be used here if parallelized. Some examples include ParamILS [Hutter *et al.*, 2007] and SMAC [Hutter *et al.*, 2011].

We describe and test two sample algorithms. The first algorithm we tested is random search [Bergstra and Bengio, 2012]. This is notably similar to a screening design, where random combinations are chosen and tested, effectively ignoring any learning potential offered by the iterative process.

Algorithm 1: SMBO(N, model, f). The SMBO algorithm.

input : The number of iterations to perform N, a surrogate model, and an expensive function f.
output: A list of combinations to try.

$dataset \leftarrow \emptyset$
for $i \leftarrow 1$ **in** *range(N)* **do**
  model.fit($dataset$)
  $x' \leftarrow \arg\max_{x\in\mathcal{X}}$ model.predict(x)
  $dataset \leftarrow dataset \cup f(x')$
end
Return $dataset$

It will be used as a baseline for comparison of other algorithms. The second algorithm augments random search by oversampling the search space by a factor c and then filtering the samples using a random forest regression.

5 Evaluation

Evaluation of the performance of the algorithms used in the workflow requires the ability to fetch the true performance of a combination. Although we do not currently have a compelete dataset to test upon, there are incomplete datasets available.

5.1 Dataset Details

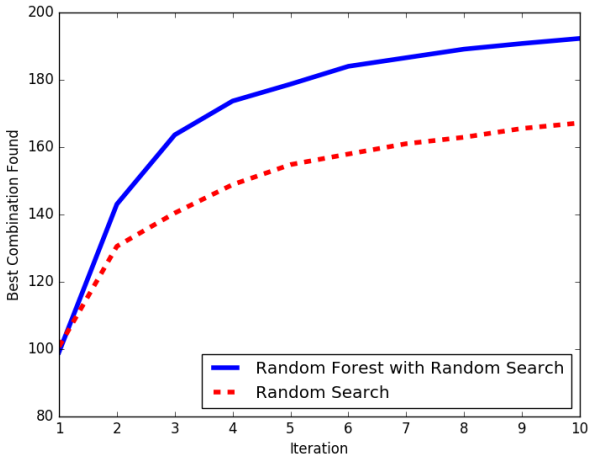
We use a dataset which describes the multi-enzyme pathway for the production of violacein [Lee *et al.*, 2013] by optimizing promoters. Promoters are genetic elements that can be represented as blocks. The pathway is constructed using five promoters. This translates into a design with five list blocks. In this dataset, there are five promoters, enumerated one through five. These promoters are characterized and enumerated such that Promoter1 is the strongest and Promoter5 is the weakest. These promoters can appear multiple times in this design and each of the list blocks can take on the sequence of any of the promoters. The size of the combinatorial space is therefore $5^5 = 3125$, as it is a 5×5 design. The design for this dataset is shown in Figure 1. The dataset itself contains approximately 200 unique combinations, with multiple measurements for several variants of violacein. We focus on the first measurement of violacein. The maximum performance in this data set is 212.5.

To use this sparse dataset for the evaluation of combinatorial workflows, the dataset must be extended to contain values for the entire combinatorial space. To do this, we train a surrogate model to estimate points which have no value in the dataset. This surrogate model will be given additional metadata available in the dataset, which will not be available to the combinatorial design algorithm. We provide this information to the model by using the numerical values for the promoters. To prevent giving this information to the combinatorial design algorithm, we encode the values as a one-hot vector, which is a method for encoding categorical variables as a vector. A one-hot vector for a single category is a binary vector where the dimension is the number of options in the category. All the dimensions are 0 except for the chosen option, which is 1. This can be extended to multiple categories by concatenating their respective one-hot vectors. We use a random forest regression, which has a R^2 score of 0.722 using 5-fold cross-validation. Observing the individual fold scores, the model does well for areas of the space that are near zero. However, the important parts of the space are the areas of high performance and the model struggles to predict those values. Those values tend to be higher than the rest, but do not reach the same magnitude as the actual values.

5.2 Results

We compare random search with an augmented version that utilizes random forest. We show the performance of the best combination found up to that point.

Figure 3: **Best Performance Found at each Iteration.** Figure depicts the average performance of the best combination found across all prior iterations in arbitrary units. We sample 100 complete rounds for both random search and the random search augmented with a random forest. We test over 10 iterations with 20 combinations tested per iteration for a total of 200 combinations tested.



In figure 3, we see the rate of improvement of each algorithm over the number of iterations. The random forest has a moderate improvement over the random search. We hypothesize that a greater improvement would be seen in a larger combinatorial space, where large regions of the space cannot be simply covered with a random search. In figure 4, we see little improvement past 25 iterations. This is likely due to the algorithm having no local search properties. An algorithm that searches through local space would be able to leverage the additional information gathered in each iteration. The random search curve, shown in Figure 4 as a control, is relatively flat, as would be expected as it does not have any learning properties.

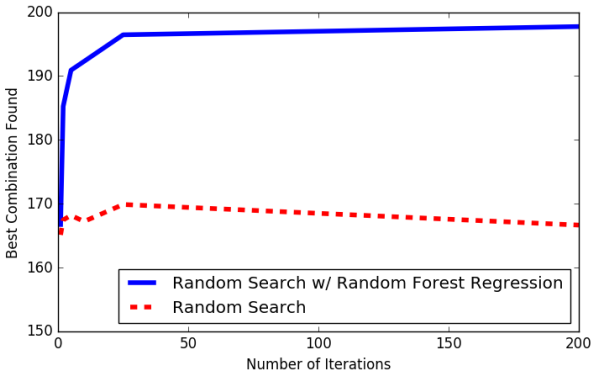
6 Conclusion

We designed a tool to optimize genetic systems by using an iterated combinatorial design workflow, which is enabled by recent progress in DNA synthesis and construction technologies. A design specification is presented, along with requirements for an algorithm to decide which combinations should be prioritized. We present a simplistic algorithm to demonstrate the method.

Future work will require well-developed datasets. These datasets should cover the full combinatorial space, as to avoid having to use a surrogate model during evaluation. The space should be sufficiently large such that it simulates real-world designs and exploration of the space becomes necessary. The design utilized for such a dataset should also incorporate a wider variety of parts and preferably non-linear metabolic pathways.

Future algorithms for suggesting higher performing combinations are also required. Future algorithms may be able to handle multiple objectives, have a local search mechanism, and will be able to utilize additional, sparse metadata. They may also be able to leverage biological details, such as incorporating sequence of parts or the type of the part. For example, we may be able to reduce all promoters and terminators in the sequence into coefficients for a gene's effect on the system by using a dynamical model. This may effectively act as a feature selection step.

Figure 4: **Performance vs Iterations, normalized for 200 combinations tested.** Figure depicts the average final performance of each algorithm tested with various configurations of batch sizes and number of iterations. The configurations tested are normalized for the number of combinations tested to show the increase of performance when given more iterations, where the data point at 200 iterations represent a sequential process. This displays the algorithms ability to utilize the information provided by the iterative process.



User input and suggestion to the algorithms may also be a desirable feature. This poses several challenges; the algorithms must be able to respond to the user's input, which can e considered as another objective in a multi-objective function. Additionally, a simple, flexible user interface must also be developed. To further increase the flexibility offered to the user, more general combinatorial designs may be permitted. This may include directionality of parts, or removing the permutation invariance assumption.

Acknowledgements

This work was funded entirely by Autodesk. The authors thank Morgan Price in the ArkinLab of Lawrence Berkeley National Laboratory for his review of this article, the Duber Laboratory at the University of California Berkeley for the violacein data, and the Bio/Nano Research Group at Autodesk for their continued support.

References

[Bergstra and Bengio, 2012] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[Bergstra *et al.*, 2011] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[Hutter *et al.*, 2007] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proc. of the Twenty-Second Conference on Artifical Intelligence (AAAI '07)*, pages 1152–1157, 2007.

[Hutter *et al.*, 2011] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, page 507523, 2011.

[Keogh and Mueen, 2010] Eamonn Keogh and Abdullah Mueen. *Encyclopedia of Machine Learning*, chapter Curse of Dimensionality, pages 257–258. Springer US, Boston, MA, 2010.

[Lee *et al.*, 2013] Michael E Lee, Anil Aswani, Audrey S Han, Claire J Tomlin, and John E Dueber. Expression-level optimization of a multi-enzyme pathway in the absence of a high-throughput assay. *Nucleic acids research*, page gkt809, 2013.

[Smanski *et al.*, 2014] Michael J Smanski, Swapnil Bhatia, Dehua Zhao, YongJin Park, Lauren BA Woodruff, Georgia Giannoukos, Dawn Ciulla, Michele Busby, Johnathan Calderon, Robert Nicol, et al. Functional optimization of gene clusters by combinatorial design and assembly. *Nature biotechnology*, 32(12):1241–1249, 2014.

[Wilson *et al.*, 0] Erin H. Wilson, Shiori Sagawa, James W. Weis, Max G. Schubert, Michael Bissell, Brian Hawthorne, Christopher D Reeves, Jed Dean, and Darren Platt. Genotype specification language. *ACS Synthetic Biology*, 0(0):null, 0. PMID: 26886161.