



7th INTERNATIONAL WORKSHOP ON BIO-DESIGN AUTOMATION
UNIVERSITY OF WASHINGTON
AUGUST 19-21, 2015

FOREWORD

Welcome to IWBDA 2015!

The IWBDA 2015 Executive Committee welcomes you to Seattle, Washington for the Seventh International Workshop on Bio-Design Automation (IWBDA) at the University of Washington. IWBDA brings together researchers from the synthetic biology, systems biology and design automation communities. The focus is on concepts, methodologies and software tools for the computational analysis and synthesis of biological systems.

The field of synthetic biology, still in its early stages, has largely been driven by experimental expertise, and much of its success can be attributed to the skill of researchers in specific domains of biology. Although there has been a concerted effort to assemble repositories of standardized components, creating and integrating synthetic components remains an ad hoc process. Inspired by these challenges, the field has seen a proliferation of efforts to create computer-aided design tools addressing synthetic biology's specific design needs – many drawing on prior expertise from the electronic design automation (EDA) community. IWBDA offers a forum for cross-disciplinary discussion, with the aim of seeding and fostering collaboration between the biological and the design automation research communities.

IWBDA is proudly organized by the non-profit Bio-Design Automation Consortium (BDAC). BDAC is an officially recognized 501(c)(3) tax-exempt organization.

This year the program consists of 17 contributed talks and 16 poster presentations. Talks are organized into five sessions: Standards & Data Exchange; Pathways and Oligo Design; Process Management; Genetic Circuits I; and Genetic Circuits II. In addition, we are very pleased to have two distinguished invited speakers: Dr. Miriah Meyer from the University of Utah and Dr. Eric Klavins from the University of Washington.

We thank all the participants for contributing to IWBDA; we thank the Program Committee for reviewing abstracts; and we thank everyone on the Executive Committee for their time and dedication. Finally, we thank National Science Foundation (NSF), Synthetic Biology Engineering Research Center (Synberc), Autodesk, Twist Bioscience, ACS Synthetic Biology, Raytheon BBN Technologies, Minres Technologies, Riffyn, Cytoscape and Lattice Automation for their support.

THE FOLLOWING PARTICIPANTS WERE PROVIDED FINANCIAL SUPPORT BY OUR SPONSORS TO ATTEND IWBDA 2015

Aaron Heuckroth	Boston University
Anuva Kulkarni	Carnegie Mellon University
Charles Fracchia	MIT Media Lab
Curtis Madsen	New Castle University
Evan Appleton	Boston University
Faisal Reza	Yale University
Gleb Kuznetsov	Harvard University
Haiyao Huang	Boston University
Hasan Baig	Technical University of Denmark (DTU)
Jacob Becraft	MIT
Jenhan Tao	University of California, San Diego
Kathleen Lewis	Boston University
Kim de Mora	iGEM HQ
Leandro Watanabe	University of Utah
Linh Huynh	University of California, Davis
Martín Gutiérrez	Universidad Politécnica de Madrid
Michael Quintin	Boston University
Navneet Rai	University of California, Davis
Nicholas Roehner	Boston University
Owen Gilfellon	New Castle University
Prashant Vaidyanathan	Boston University
Ryan Silva	Boston University
Swapnil Bhatia	Boston University
Swati Carr	Boston University
Tasuku Kitada	MIT
Tim Fiori	University of Australia
Tramy Nguyen	University of Utah
Tyler Wagner	Boston University

IWBDA 2014 SPONSORS

design



workflow



class



algorithm

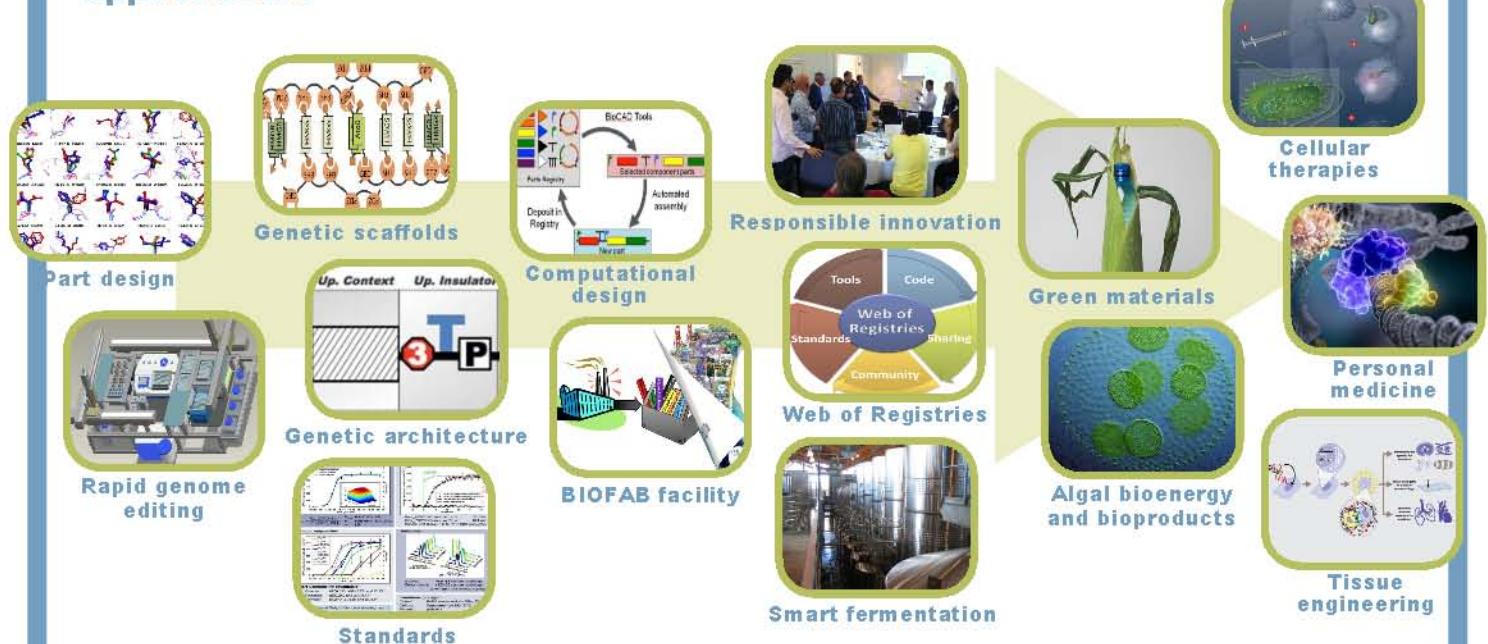




Building the Future with Biology.

A proud sponsor of IWBDA

Synberc is a major U.S. research consortium that is creating the foundational tools and technologies needed for 21st century bio-based applications:



#meetsynbio

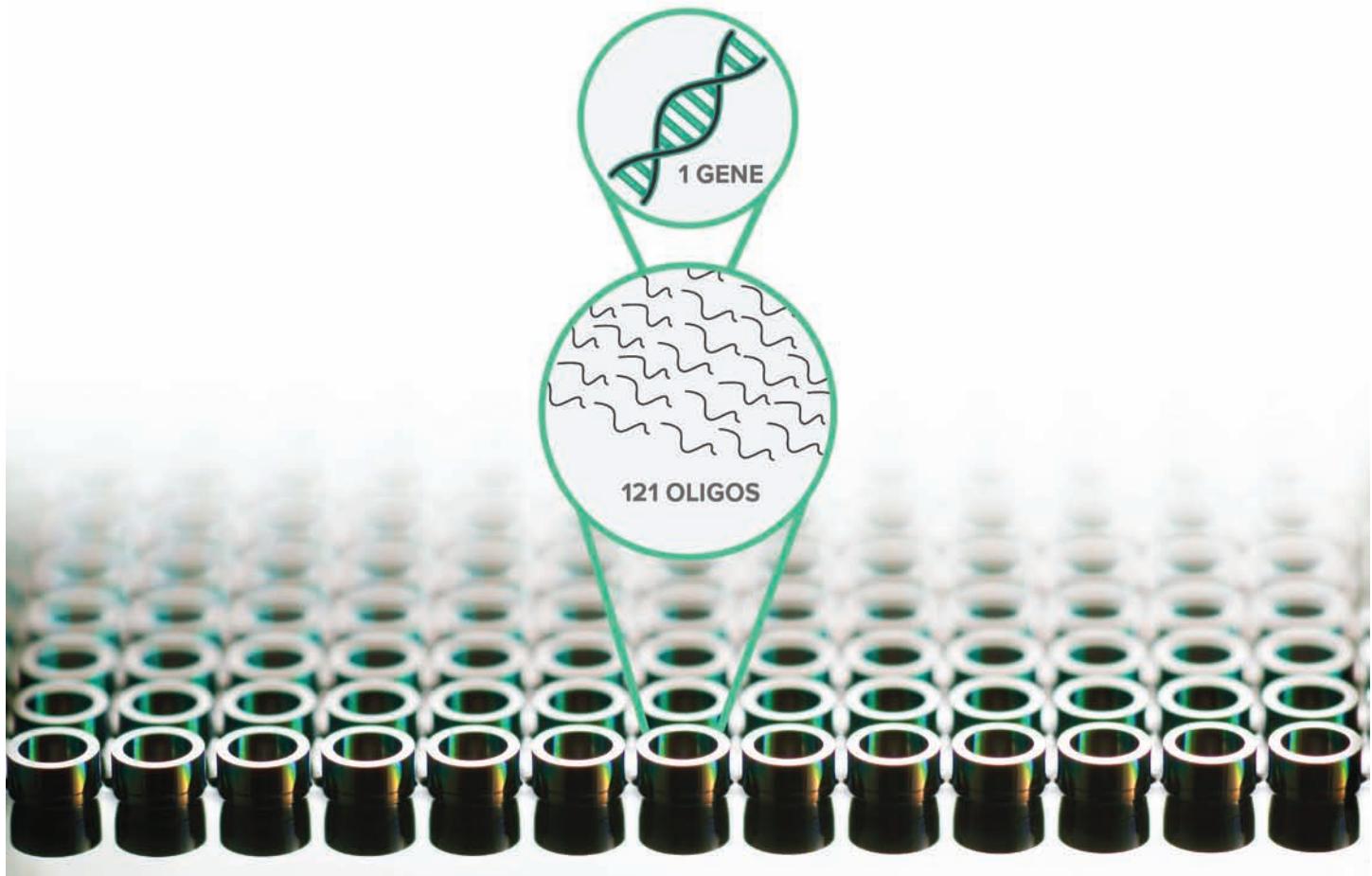
Synberc wants to help practitioners better engage with the public about why we are interested in pursuing synthetic biology and who is doing this work. So we launched the #meetsynbio project, which aims to put the real people behind synthetic biology in touch with the real people in our community whom we endeavor to benefit. Find out how you can participate at:

synberc.org/meetsynbio



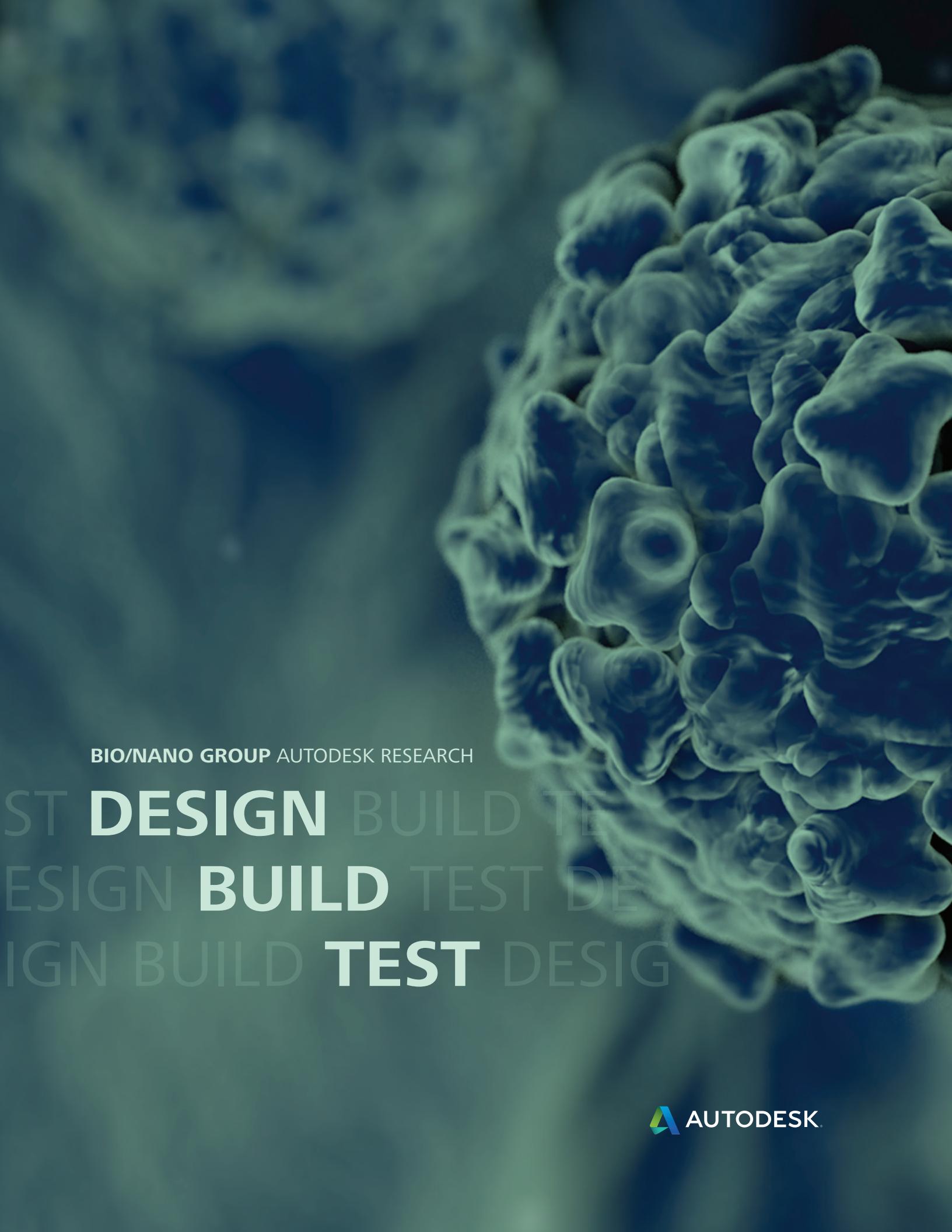
Alpha Manufacturing Program is now open

Experience the Power of Scale



Just the start of something great

Learn more: sales@twistbioscience.com



BIO/NANO GROUP AUTODESK RESEARCH

ST DESIGN BUILD TEST
DESIGN BUILD TEST DE
IGN BUILD TEST DESIG

CHALLENGE CONVENTION BE PART OF THE NEXT FIRST.

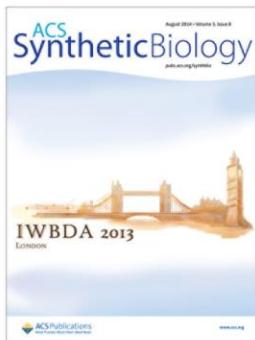


Raytheon BBN Technologies has been providing advanced technology research and development for more than six decades. From the ARPANET and the first email, to GenBank and the first digital stereo mammography system, through the first network protected by quantum cryptography and our lifesaving Boomerang gunshot detection technology, BBN has consistently transitioned advanced research into innovative, practical solutions. Today, BBN scientists and engineers continue to take risks and challenge convention to create new and fundamentally better solutions.

<http://jobs.raytheon.com/search/?keyword=BBN>

Raytheon
BBN Technologies

Submit your research to the *ACS Synthetic Biology* 'IWBDA 2015' Special Issue



Led by Editor-in-Chief Christopher A. Voigt, *ACS Synthetic Biology* provides a high quality forum for the best research in synthetic biology and systems bioscience.

We would like to invite all attendees to submit original work to our '**IWBDA 2015' Special Issue**'. *ACS Synthetic Biology* has the highest editorial standards, offers rapid publication of research findings and imposes NO author submission, page, color, or cover art charges. We are looking for high quality submissions, for peer-review, as either short Letters or full-length Articles.

The submission deadline is September 30, 2015.

ACS
Synthetic Biology

pubs.acs.org/acssyntheticbiology

Oral Presentation Abstracts – Table of Contents

Progress from the Synthetic Biology Standards Consortium	18
<i>Matthew Munson, Sarah Munro and Marc Salit.</i>	
The Synthetic Biology Open Language 2.0	20
<i>Bryan Bartley, Jacob Beal, Kevin Clancy, Goksel Misirlı, Nicholas Roehner, Matthew Pocock, Tramy Nguyen, Zhen Zhang, Chris Myers, John Gennari, Herbert Sauro, Curtis Madsen, Anil Wipat and Ernst Oberortner</i>	
A Converter from the Systems Biology Markup Language to the Synthetic Biology Open Language	22
<i>Tramy Nguyen and Chris Myers</i>	
SBOL Stack: The One-stop-shop for Storing and Publishing Synthetic Biology Designs	24
<i>Curtis Madsen, Goksel Misirlı, Matthew Pocock, Jennifer Hallinan and Anil Wipat.</i>	
Stoichiometrically Minimal Source Pathways via Model Checking.	26
<i>Matthew Fong and Sanjit Seshia.</i>	
Double Dutch: A Tool for Designing Libraries of Variant Metabolic Pathways.	28
<i>Nicholas Roehner and Douglas Densmore.</i>	
Millstone: Software for iterative genome engineering	30
<i>Gleb Kuznetsov, Daniel B. Goodman, Marc J. Lajoie, George M. Church, Kevin Y. Chen, Changping Chen, Michael G. Napolitano and Brian W. Ahern</i>	
MERLIN: a DNA Design Tool for Large-Scale Genome Engineering	32
<i>Michael Quintin, Aaron Lewis, Natalie Ma, Douglas Densmore and Farren Isaacs</i>	
Software for Engineering Biology in a Multi-Purpose Foundry	34
<i>Benjie Chen, Dan Cahoon, Barry Canton and Austin Che</i>	
YeastFabCAM: Computer Assisted Manufacturing for constructing large-scale genetic parts registries.	36
<i>Yisha Luo, Yue Shen, Emily Scher, Junbiao Dai and Yizhi Cai</i>	
Successful Failure: Best Practices for Quality Control of Large-scale DNA Assembly	38
<i>Bryan Bartley, Michal Galdziński, John Gennari and Herbert Sauro</i>	
Automated design of genetic logic circuits	40
<i>Bryan Der, Douglas Densmore and Christopher Voigt</i>	
Parameter inference for gene circuit models	42
<i>Linh Huynh and Ilias Tagkopoulos.</i>	
Design of Biological Circuits Using Signal-to-Noise Ratio.	44
<i>Jacob Beal</i>	
SynBad: An SVP Design Framework.	46
<i>Owen Gilfellon, Goksel Misirlı, Curtis Madsen, Jennifer Hallinan, Paolo Zuliani and Anil Wipat</i>	
D-VASim: Dynamic Virtual Analyzer and Simulator for Genetic Circuits	48
<i>Hasan Baig and Jan Madsen</i>	
Fluigi: An Automated Framework for Creating Bioelectronic Devices	50
<i>Haiyao Huang, Aaron Heuckroth, Ryan J. Silva and Douglas Densmore</i>	

Poster Presentation Abstracts – Table of Contents

archiYEAST: a command-line synthetic yeast architect.....	52
<i>Laura Adam and Eric Klavins.</i>	
Big Mechanism Design and Analysis Automation	54
<i>Anuva Kulkarni, Cheryl Telmer, and Natasa Miskov-Zivanov.</i>	
On the complexity of codon context optimization	56
<i>Dimitris Papamichail, Hongmei Liu, and Georgios Papamichail.</i>	
Context Aware Pipetting.	58
<i>Charles Fracchia, Joseph Jacobson, and George Church.</i>	
CRISPR and TAL Search and Design Tools for Cell Engineering	60
<i>Daniel Williams, Sridhar Ranganathan, and Joel Brockman.</i>	
Design and Characterization of Genetic Circuits using Multiplex DNA Synthesis.....	62
<i>Daniel Goodman, Casper Enghuus, and George Church.</i>	
Design optimizations of precise synthetic genome targeting and editing small molecules for diverse disease loci.....	64
<i>Faisal Reza and Peter M. Glazer.</i>	
A Detailed, flexible standard for sharing DNA concepts.....	66
<i>Barbara Frewen, Jed Dean, and Aaron Kimball.</i>	
Efficient Analysis of SBML Models Using Arrays.....	68
<i>Leandro Watanabe and Chris Myers.</i>	
Extending the features and improving the performance of gro simulator: new bacterial conjugation and gene expression modules.....	70
<i>Martín Gutiérrez, Paula Gregorio-Godoy, Guillermo Pérez Del Pulgar, and Alfonso Rodríguez-Patón.</i>	
Phagebook: A Software Environment for Social Synthetic Biology.	72
<i>Kathleen Lewis, Inna Turshudzhyan, Kara Le Fort, Nicholas Musella, Nicholas Roehner, Prashant Vaidyanathan, and Douglas Densmore.</i>	
Phoenix: An automated design-build-test tool.	74
<i>Evan Appleton, Yash Agarwal, Zachary Chapasko, Ernst Oberortner, Alan Pacheco, Prashant Vaidyanathan, Nicholas Roehner, and Douglas Densmore.</i>	
Pooled, in situ assembly of complex genomic libraries using sorter-assisted genome engineering.....	76
<i>Robert Egbert, Eric Yu, and Adam Arkin.</i>	
Scylax™ - Automated design of cell factories incorporating synthetic enzymes.....	78
<i>Michał Galdzicki, Kyle Medley, Rudesh D Toofanny, Stanley Gu, Yih-En Andrew Ban, Herbert M Sauro, and Alexandre Zanghellini.</i>	
Towards Semi-Automated Experimental Design Using Model Inference in Synthetic Biology. 80	
<i>Tileli Amimeur.</i>	
Towards a sequence-level DNA design specification language.	82
<i>Nicholas Bolten and Eric Klavins.</i>	

Organizing Committee

Executive Committee

General Chair – Douglas Densmore (Boston University)

Finance Chair – Traci Haddock (iGEM HQ)

Program Committee Chairs – Jacob Beal (BBN Technologies)

Publication Chair – Swati Carr (Boston University)

Local Chairs – Eric Klavins (University of Washington) and

Traci Haddock (iGEM HQ)

Web Chair - Aaron Adler (BBN Technologies)

Bio-Design Automation Consortium

Douglas Densmore (Boston University), President

Aaron Adler (BBN Technologies), Vice-President

Traci Haddock (iGEM HQ), Treasurer

Natasa Miskov-Zivanov (Carnegie Mellon University), Clerk

Founders

Douglas Densmore (Boston University)

Soha Hassoun (Tufts University)

Marc Riedel (University of Minnesota)

Program Committee

Shota Atsumi, University of California, Davis

Swapnil Bhatia, Boston University

Bryan Der, Massachusetts Institute of Technology

Barbara Di Ventura, University of Heidelberg

Michal Galdzicki, Arzeda Corp.

Soha Hassoun, Tufts University

Nathan Hillson, Joint BioEnergy Institute

Natasa Miskov-Zivanov, Carnegie Mellon University

Chris Myers, University of Utah

Dimitris Papamichail, The College of New Jersey

Cesar Rodriguez, Autodesk

Nicholas Roehner, Boston University

Herbert Sauro, University of Washington

Guy-Bart Stan, Imperial College London

Darko Stefanovic, University of New Mexico

Ilias Tagkopoulos, University of California, Davis

Sean Ward, Synthace

IWBDA 2015 program

Thursday, August 20th

- 8:00 - 8:30 Breakfast and Registration
8:30 - 8:40 Opening Remarks, Douglas Densmore, BDAC General Chair

Talk Session I: Standards & Data Exchange, Moderator: Aaron Adler

- 8:40 - 9:00 *Progress from the Synthetic Biology Standards Consortium*
Matthew Munson, Sarah Munro and Marc Salit
9:00 - 9:20 *The Synthetic Biology Open Language 2.0*
Bryan Bartley, Jacob Beal, Kevin Clancy, Goksel Misirli, Nicholas Roehner, Matthew Pocock, Tramy Nguyen, Zhen Zhang, Chris Myers, John Gennari, Herbert Sauro, Curtis Madsen, Anil Wipat and Ernst Oberortner
9:20 - 9:40 *A Converter from the Systems Biology Markup Language to the Synthetic Biology Open Language*
Tramy Nguyen and Chris Myers
9:40 - 10:00 *SBOL Stack: The One-stop-shop for Storing and Publishing Synthetic Biology Designs*
Curtis Madsen, Goksel Misirli, Matthew Pocock, Jennifer Hallinan and Anil Wipat
10:00 - 10:30 Coffee Break

Keynote

- 10:30 - 11:30 **Eric Klavins, University of Washington.** Programmable Synthetic Biology.

Discussion Session I, Leader: Evan Appleton

- 11:30 - 12:30 Topic: Cloud Labs vs. Desktop Robots: Can there be only one?

Lunch

- 12:00 - 12:30

Poster Session & Demos

- 1:00 - 2:30

Talk Session II: Pathway and Oligo Design, Moderator: Natasa Miskov-Zivanov

- 2:30 - 2:50 *Stoichiometrically Minimal Source Pathways via Model Checking*
Matthew Fong and Sanjit Seshia
2:50 - 3:10 *Double Dutch: A Tool for Designing Libraries of Variant Metabolic Pathways*
Nicholas Roehner and Douglas Densmore
3:10 - 3:30 *Millstone: Software for iterative genome engineering*
Gleb Kuznetsov, Daniel B. Goodman, Marc J. Lajoie, George M. Church, Kevin Y. Chen, Changping Chen, Michael G. Napolitano and Brian W. Ahern
3:30 - 4:00 Coffee Break

Lab Tour

- 4:00 - 5:00 *Klavins Lab Tour*

Evening Activities

- 7:00 - 9:00 Dinner at UW Club

IWBDA 2015 program

Friday, August 21st

8:00 - 8:30 Breakfast and Registration

Talk Session III: Process Management, Moderator: Nic Roehner

- 8:30 - 8:50 *MERLIN: a DNA Design Tool for Large-Scale Genome Engineering*
Michael Quintin, Aaron Lewis, Natalie Ma, Douglas Densmore and Farren Isaacs
- 8:50 - 9:10 *Software for Engineering Biology in a Multi-Purpose Foundry*
Benjie Chen, Dan Cahoon, Barry Canton and Austin Che
- 9:10 - 9:30 *YeastFabCAM: Computer Assisted Manufacturing for constructing large-scale genetic parts registries*
Yisha Luo, Yue Shen, Emily Scher, Junbiao Dai and Yizhi Cai
- 9:30 - 9:50 *Successful Failure: Best Practices for Quality Control of Large-scale DNA Assembly*
Bryan Bartley, Michal Galdzicki, John Gennari and Herbert Sauro
- 9:50 - 10:20 Coffee Break

Industry Showcase, Moderator: Jake Beal

- 10:20 - 10:40 *Realizing the dream of clean, structured data: Bringing manufacturing-grade quality to R&D*, Timothy Gardner (Riffyn)
- 10:40 - 11:00 Chris Grant (Synthace)

Keynote

- 11:00 - 12:00 **Miriah Meyer, University of Utah.** Why an (interactive) visualization is worth a thousand numbers.

Lunch

12:00 - 1:30

Discussion Session II, Leader: Doug Densmore

- 1:30 - 2:30 Topic: BDA Commercialization Business Models

Talk Session IV: Genetic Circuits I, Moderator: Traci Haddock

- 2:30 - 2:50 *Automated design of genetic logic circuits*
Bryan Der, Douglas Densmore and Christopher Voigt
- 2:50 - 3:10 *Parameter inference for gene circuit models*
Linh Huynh and Ilias Tagkopoulos
- 3:10 - 3:30 *Design of Biological Circuits Using Signal-to-Noise Ratio*
Jacob Beal
- 3:30 - 4:00 Coffee Break

Talk Session V: Genetic Circuits II, Moderator: Kevin LeShane

- 4:00 - 4:20 *SynBad: An SVP Design Framework*
Owen Gilfellon, Goksel Misirli, Curtis Madsen, Jennifer Hallinan, Paolo Zuliani and Anil Wipat
- 4:20 - 4:40 *D-VASim: Dynamic Virtual Analyzer and Simulator for Genetic Circuits*
Hasan Baig and Jan Madsen
- 4:40 - 5:00 *Fluigi: An Automated Framework for Creating Bioelectronic Devices*
Haiyao Huang, Aaron Heuckroth, Ryan J. Silva and Douglas Densmore

Closing Remarks

- 5:00 - 5:15 Doug Densmore

Keynote Presentation

Eric Klavins

Talk Title: Programmable Synthetic Biology



Dr. Eric Klavins is an associate professor of electrical engineering at the University of Washington in Seattle. He received a B.M. in Music in 1992 and a B.S. in computer science in 1996 from San Francisco State University. He received the M.S. and Ph.D. degrees in computer science and engineering in 1999 and 2001 from the University of Michigan, Ann Arbor. From 2001 to 2003 he was a postdoctoral scholar in the Control and Dynamical Systems Department at the California Institute of Technology where he worked with Richard Murray. In 2003 Eric was hired in Electrical Engineering at the University of Washington in Seattle, WA and received tenure in 2009. He holds adjunct appointments in Computer Science and Engineering and in Bioengineering and is the Director for the UW Center for Synthetic Biology.

Until approximately 2008, Klavins' research was primarily in computer science and control systems, focusing on stochastic processes, robotics and self-assembly. At about this time, he learned the basics of genetic engineering of the next few years switched entirely fields to synthetic biology and now runs an interdisciplinary group of engineers, biologists, experimentalists, and theorists -- all focused on engineering life. His current projects include synthetic multicellular systems with engineered bacteria and yeast, modeling and design for synthetic multicellular systems, and laboratory automation.

Keynote Presentation

Miriah Meyer

Talk Title: Why an (interactive) visualization is worth a thousand numbers.



Dr. Miriah Meyer is a USTAR assistant professor in the School of Computing at the University of Utah and a faculty member in the Scientific Computing and Imaging Institute. Her research focuses on the design of visualization systems for helping researchers make sense of complex data. She obtained her bachelors degree in astronomy and astrophysics at Penn State University, and earned a PhD in computer science from the University of Utah. Prior to joining the faculty at Utah Miriah was a postdoctoral research fellow at Harvard University and a visiting scientist at the Broad Institute of MIT and Harvard.

Miriah is the recipient of a NSF CAREER grant, a Microsoft Research Faculty Fellowship, and a NSF/CRA Computing Innovation Fellow award. She was named both a TED Fellow and a PopTech Science Fellow, as well as included on MIT Technology Review's TR35 list of the top young innovators and Fast Company's list of the 100 most creative people. She was also awarded an AAAS Mass Media Fellowship that landed her a stint as a science writer for the Chicago Tribune.

Allan Kuchinsky Scholarship

Swapnil Bhatia



Dr. Swapnil Bhatia is a research assistant professor at the Department of Electrical and Computer Engineering at Boston University and co-founder of Lattice Automation. He received his PhD in computer science from the University of New Hampshire in 2010. He worked as a postdoctoral scholar in the Cross-disciplinary Integration of Design Automation Research (CIDAR) laboratory with Prof. Douglas Densmore where he was a key contributor to projects funded by the DARPA Living Foundries, DARPA Synthetic Biology Seedling and the National Science Foundation (NSF). In addition to conducting research, Swapnil enjoys mentoring students and has been involved in some capacity with almost every graduate student and undergraduate that worked in the CIDAR lab during his time there.

Swapnil's research focus is the development of tools for machine learning genetic design principles from biological screening data. He led a team that created a platform for synthetic biology automation (Puppeteer); he was the lead developer of a combinatorial design software tool (Finch) and was part of the TASBE tool chain for synthetic biology developer team. He has also developed algorithms for de novo sequencing from mass spectrometry data.

The first annual Allan Kuchinsky Scholarship to IWBDA is being generously sponsored by Cytoscape.



Progress from the Synthetic Biology Standards Consortium

Matthew S. Munson

NIST/Stanford University

Joint Initiative for Metrology in Biology
443 Via Ortega, Room 225
Stanford, CA 94305
mmunson@nist.gov

Sarah A. Munro

NIST/Stanford University

Joint Initiative for Metrology in Biology
443 Via Ortega, Room 225
Stanford, CA 94305
smunro@nist.gov

Marc Salit

NIST/Stanford University

Joint Initiative for Metrology in Biology
443 Via Ortega, Room 231
Stanford, CA 94305
salit@nist.gov

ABSTRACT

The NIST-hosted Synthetic Biology Standards Consortium (SBSC) will collectively build the infrastructure to support a fully integrated global synthetic biology enterprise. We aim to accomplish this by developing metrology products – standards, including reference materials, reference data, reference methods, and documentary standards – that will enable coordination of labor and reuse of materials. We will present the results of the kick-off workshop for the SBSC, held on March 31, 2015 at Stanford University. A summary of the plans developed by each working group will be shared, and mechanisms of future consortium operations will be discussed.

1. INTRODUCTION

Synthetic biology will realize its full contributions to the bioeconomy when a robust metrology infrastructure is in place to enable coordination of labor and reuse of materials. Metrology products – standards, including reference materials, reference data, reference methods, and documentary standards – can enable business-to-business transactions at scale. The NIST-hosted Synthetic Biology Standards Consortium (SBSC) will collectively build the infrastructure to support a fully integrated global synthetic biology enterprise. [2]

The structure of the consortium is represented schematically in Figure 1. Volunteer participants come together, supported by NIST, to create metrology products. NIST provides hosting and professionalization of the standards development process. Members contribute by providing their metrology needs and technical expertise for metrology product development.

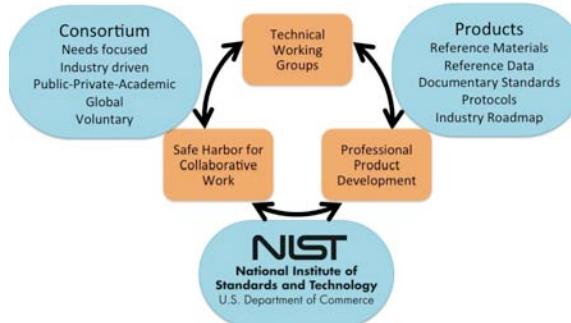


Figure 1 Consortium Structure: Volunteer participants, hosted by NIST organize to create metrology products that support the growth of the bioeconomy. NIST provides hosting and professionalization of Standards Development. Members provide metrology needs and labor sharing.

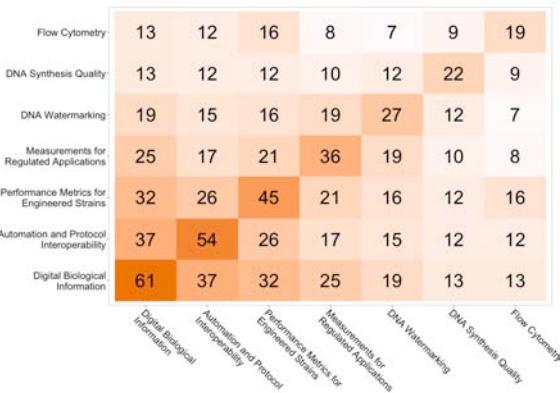


Figure 2 Registration and Working Group Interest: 157 people registered, with 123 people attending (110 in person and 13 remotely). The break down and overlap of their interests in the various working groups is shown. Volunteers were recruited to lead panel and working group discussions for each topic.

Consortium direction and decision making is consensus-based and data-driven. The SBSC has been convened with a kickoff workshop on March 31, 2015. [1,3] The goal of the workshop was to identify several initial working groups with critical mass, leadership teams, and a clear path forward to deliver standards.

2. WORKSHOP STRUCTURE

In order to achieve this aim, we structured the workshop using panel discussions and break out groups. The panels were composed of volunteers from the community who, at the time of registration, indicated an interest in leading the efforts in a candidate technical area (interest in the candidate working groups is shown in Figure 2).

These participants were then solicited to speak to the consortium as part of a panel. Each panelist was asked to shape their remarks by considering three questions:

- What problem will this working group solve?
- Who needs this problem solved?
- What products will you develop together to solve the problem? What will success look like?

3. WORKSHOP OUTPUTS

Groups formed to discuss these questions and build consensus on terms of reference for each group in the following technical areas: Automation and Protocol Interoperability, Flow Cytometry, Digital Biological Information, DNA Construction, Measurements for Regulated Applications, and Performance Metrics for Engineered Systems. Graphical representation of the terms of reference for each group is shown in Figure 3. A workshop summary report has been produced. [4]

Moving forward, NIST will continue to coordinate the efforts of each group to refine their terms of reference and develop initial metrology products. This facilitation will take the form of hosting teleconferences, guiding development of metrology products, and providing coordination of labor. Consortium membership remains free and open; we solicit the engagement and contributions of all interested parties.

4. REFERENCES

- [1] Cavanagh R. 2015. "Synthetic Biology Standards Consortium – Kick-off Workshop" Federal Register, Vol. 80, No. 56; pp 15563-15564.
- [2] Galdzicki, M., Munro, S.A., Boyle, P., and Ubersax J. 2012. "A Vision for a Synthetic Biology Standards Consortium" <http://synbioleap.org/wp-content/uploads/2013/05/a-vision-for-a-synthetic-biology-standards-consortium.pdf>.
- [3] Hayden ,E.C. 2015. Synthetic biology called to order. *Nature*, 520, (April 9, 2015), 141-142. DOI= <http://dx.doi.org/10.1038/520141a>.
- [4] Munson, M.S., Munro, S.A., and Salit, M. 2015. Synthetic Biology Standards Consortium Workshop Report. http://jimb.squarespace.com/s/SBSC_Workshop_Summary_Report.pdf.

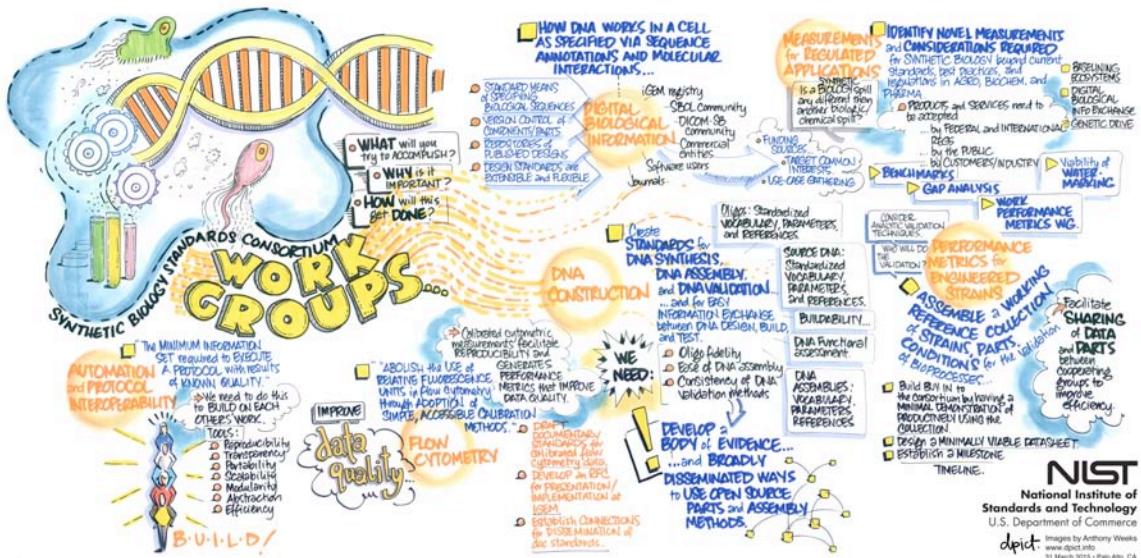


Figure 3 Output from the working group discussions: Groups spent 90 minutes developing terms of reference for their efforts to answer; "What will we do?"; "Why is it important?"; and "How will we accomplish it?". The groups reported back to the consortium as a whole. These reports are summarized graphically, above.

The Synthetic Biology Open Language 2.0

Bryan Bartley
University of Washington, US
bbartley@uw.edu

Goksel Misirli
Newcastle University, UK
goksel.misirli@ncl.ac.uk

Jacob Beal
Raytheon BBN Tech., US
Jakebeal@bbn.com

Nicholas Roehner
Boston University, US
nroehner@bu.edu

Kevin Clancy
ThermoFisher Scientific, US
kevin.clancy1@thermofisher.com

Matthew Pocock
Turing Ate My Hamster, UK
turingatemymhamster@gmail.com

1. INTRODUCTION

The initial version of the Synthetic Biology Open Language (SBOL) was designed for the exchange of information about biological designs at the DNA level. As the field of synthetic biology matures, however, there is a clear need to extend SBOL to capture the function of biological designs and their structure beyond annotated DNA sequences [2]. To support the specification of increasingly complex and diverse biological designs, standards need to represent data on both biological structure and function in a modular, hierarchical fashion. These include data on biological interactions, which are especially important for the functional composition of biological components, and meta-data on computational models, which are important for linking biological designs to more detailed descriptions of their behavior in specific biological contexts.

SBOL 1.1 provides entities to represent biological systems as composite DNA designs [1]. In particular, biological parts are represented in SBOL 1.1 using `DnaComponent` entities. These entities can be reused in different designs, constituting building blocks of larger and more complex `DnaComponent` entities.

SBOL 2.0 builds conceptually upon the DNA-centric SBOL 1.1 data model in two directions. First, SBOL 2.0 generalizes the concept of a DNA component to support a wide range of biological components, including RNA, proteins, and metabolites. This generalization enables the structural diversity of biological designs to be fully captured. Second, SBOL 2.0 introduces a functional data model to complement its structural data model, thereby enabling specification of the dynamic interactions and processes of a design in a lightweight manner, without commitment to any specific quantitative modeling framework. Ultimately, SBOL 2.0 provides a system of hierarchical constructs for describing both the structure and function of modular biological designs.

2. SBOL 2.0 DATA MODEL

As shown in Figure 1, SBOL 2.0 offers a rich set of design entities, including `ComponentDefinitions`, `Sequences`, `ModuleDefinitions`, `Models`, `Collections` (not shown), and `GenericTopLevels` (not shown). These entities enable the design of biological systems from composable, modular, and reusable building blocks. Examples can be found in the SBOL 2.0 specification, online at <http://sbolstandard.org/>.

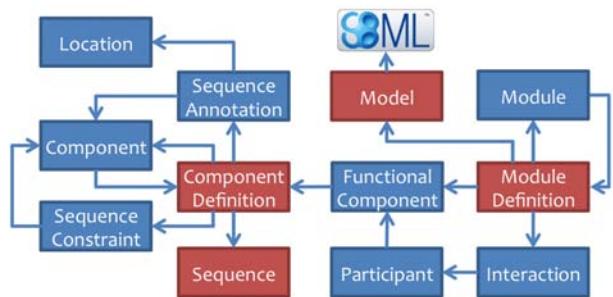


Figure 1: Red boxes represent the top level entities that may encapsulate entities represented in blue boxes. Arrows indicate property relationships (unlabeled for simplicity). The left half of the diagram is a generalization of SBOL 1.1 to include molecules other than DNA, while the right half is entirely new.

Component Definitions. Biological components are represented in SBOL 2.0 using the `ComponentDefinition` entity, which provides an improved representation of component compositions and their associated structural constraints. In SBOL 1.1, sub-components are represented via `SequenceAnnotations`. However, this representation requires even small regions of DNA, such as start codons, to be defined as reusable components. SBOL 2.0 `SequenceAnnotations`, on the other hand, simply indicate regions of interest that can refer to sub-components if desired. These sub-components are represented by `Component` entities. Furthermore, additional entities are introduced to represent different types of `Locations` for `SequenceAnnotations`, such as a cuts between adjacent base pairs and ranges. As in SBOL 1.1, SBOL 2.0 also supports the representation of partial designs, in which precise locations may not be known. Rather than use `SequenceAnnotations` to explicitly encode sub-component ordering, SBOL 2.0 represents this and other biological structural relationships between sub-components using `SequenceConstraint` entities.

Beyond DNA, the `ComponentDefinition` entity of SBOL 2.0 can also be used to represent different types of biological entities, such as RNA, protein, metabolites, small molecules, and complexes. The types and roles of these entities reference existing data in the form of terms from ontologies. For example, the roles of a `ComponentDefinition` can define whether it is a promoter or coding sequence by referring to terms from the Sequence Ontology.

Sequences. In SBOL 2.0, more general sequence information can be attached to different types of **ComponentDefinitions**. The International Union of Pure and Applied Chemistry (IUPAC) encodings are used to specify the nucleotide and amino acid **Sequences** of DNA, RNA and protein components. The Simplified Molecular Input Line Entry System (SMILES) encoding is recommended to specify the **Sequences** (atomic structures) of small molecules.

Module Definitions. A **ModuleDefinition** entity can be used to link several entities to represent the function of a biological system design. Each **ModuleDefinition** includes **FunctionalComponents**, which are defined by **ComponentDefinitions**, and the **Interactions** between these components. Information about **Interactions** is crucial to specify the qualitative functional details of a design. Each **Interaction** has one or more **Participations** that elaborate on the roles of participant **FunctionalComponents**.

Each **ModuleDefinition** can also indicate its inputs and outputs, thereby informing its composition and reuse by parent entities. For example, a parent **ModuleDefinition** can import other **ModuleDefinitions** as **Modules** and map the inputs/outputs of these sub-modules to its own. This approach aids machine reasoning and automation to compose modules into designs for complex biological systems.

Models. **Model** entities document references to actual sources for quantitative or qualitative models. Each model entity includes the model source, framework, and language. Although Figure 1 shows an SBML model linked to a **Model** entity, it is important to note that the model can be encoded in any language, such as CellML, Matlab, etc.

Extension via Annotations. In addition to the entities described here, SBOL provides an annotation framework for application-specific information. Namely, each entity in an SBOL file can be annotated with Resource Description Framework (RDF) properties. Furthermore, application-specific entities can be included as RDF documents. SBOL libraries make these custom annotations and documents available to tools as generic properties and **GenericTo-plevel** entities that are preserved during subsequent read and write operations.

3. SERIALIZATION AND LIBRARIES

SBOL documents are serialized using RDF, taking advantage of the rich tool ecosystem for this Semantic Web technology. Unique Uniform Resource Identifiers (URIs) identify each entity in a SBOL document. Libraries to read and write SBOL 2.0 documents are available in several languages, with ongoing support and development by the SBOL community. The Java library, libSBOLj 2.0 [3], is the most mature. This library is backwards compatible and can import SBOL 1.1 data into SBOL 2.0 data objects. Other ongoing library development efforts include Scala and C libraries.

4. CONTINUED DEVELOPMENT

Beyond the extensions added by SBOL 2.0, the SBOL standard is undergoing continuous development to represent more information about different types of biological system

designs. In some cases, there is not yet sufficient scientific consensus for effective standards development. Currently, the most pressing area for development is capturing data on biological context, such as experimental conditions, chassis, and growth media. Such information is not yet captured in the core objects of the standard, but can be encoded for testing as annotations and **GenericTopLevel** entities.

In this and other extension initiatives, SBOL uses existing standards and resources whenever possible. For example, SBOL already leverages existing ontologies for terms to define the types, roles, and other properties of entities in the SBOL data model. In general, SBOL 2.0 links to these external resources via placeholders and provides guidelines for their use with limited enforcement.

Finally, the development of SBOL is carried out openly and iteratively with the community feedback. SBOL is also now part of COMBINE, an initiative to coordinate the development of standards for computational modeling in biology, which aids in the application of best practices for the development of data standards.

5. ACKNOWLEDGMENTS

Beyond the listed authors, contributions to the SBOL standard have been made by many individuals and organizations that participate in the SBOL Developers Group. The work reported here has been partially supported by the National Science Foundation under Grant Number DBI-1356041 and DBI-1355909, and the Engineering and Physical Sciences Research Council under grant EP/J02175X/1. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of our funding agencies.

6. ADDITIONAL AUTHORS

Tramy Nguyen, Zhen Zhang, Chris Myers (U. of Utah, US, {tramy.nguyen, zhen.zhang, chris.john.myers}@utah.edu), John Gennari, Herbert Sauro (U. of Washington, US, {gennari,hsauro}@uw.edu), Curtis Madsen, Anil Wipat (Newcastle U., UK, {curtis.madsen,anil.wipat}@ncl.ac.uk), Ernst Oberortner (DOE Joint Genome Institute (JGI), US, eoberortner@lbl.gov), Michael Bissell (Amyris, Inc., US, bissell@amyris.com)

7. REFERENCES

- [1] M. Galdzicki et al. The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nature Biotechnology*, 32(6):545–550, 2014.
- [2] N. Roehner, E. Oberortner, M. Pocock, J. Beal, K. Clancy, C. Madsen, G. Misirli, A. Wipat, H. Sauro, and C. J. Myers. Proposed data model for the next version of the Synthetic Biology Open Language. *ACS Synthetic Biology*, 4(1):57–71, 2015.
- [3] Z. Zhang, T. Nguyen, N. Roehner, G. Misirli, M. Pocock, E. Oberortner, J. Beal, K. Clancy, A. Wipat, and C. Myers. libSBOLj 2.0: A Java library to support SBOL 2.0. In *IEEE SY-Bio Workshop to Address Topics in Systems and Synthetic Biology*, Dallas, US, Mar. 2015.

A Converter from the Systems Biology Markup Language to the Synthetic Biology Open Language

Tramy Nguyen

Dept. of Electrical and Computer Engineering
tramy.nguyen@utah.edu

Chris J. Myers

Dept. of Electrical and Computer Engineering
myers@ece.utah.edu

1. INTRODUCTION

Recently, Version 2.0 of the *Synthetic Biology Open Language* (SBOL) has been released to describe genetic designs [4]. In this new version of SBOL, component types are generalized (Protein, RNA, small molecules, etc.), and new features are added to incorporate behavioral and hierarchical aspects. The *Systems Biology Markup Language* (SBML) [2] is a widely used standard for describing biological behavior. SBOL and SBML serve different purposes. SBOL is intended to describe the structural design of genetic circuits and only basic qualitative behavioral aspects, while SBML's goal is to create models that can be simulated.

Despite their differences, conversion between their common elements is useful. In earlier work, a converter between SBOL to SBML has been reported [5]. This abstract describes a converter from SBML to SBOL. In particular, this converter begins with an SBML model with annotations using the *Systems Biology Ontology* (SBO) [1], and it infers the structure and qualitative functional aspects of the model to produce an SBOL data file. Both of these converters are now integrated within the *iBioSim genetic design automation* (GDA) tool being developed at the University of Utah.

2. CONVERSION FROM SBML TO SBOL

SBML includes the following core constructs: *Compartments*, *Species*, *Reactions*, *Parameters*, etc.. In *iBioSim*, genetic designs are described using species and reactions annotated using the SBO. These annotations enable species to be identified as promoters, mRNA, or proteins, reactions as degradation, complex formation, or genetic productions, and modifiers to reactions as activators or inhibitors.

Fig. 1(a) shows an example of a model constructed in *iBioSim* for a LacI inverter. This model is composed of three proteins, LacI, TetR, and GFP represented as blue rectangles. These proteins are represented as species in SBML. In addition, the model contains a promoter pLac. Promoters are also represented as species in SBML. *iBioSim* also includes high-level constructs for genetic circuits. A red arc represents repression and green arcs represent genetic production. These are represented by a reaction that is annotated with SBO terms to describe these relationships. Using the *hierarchical model composition* (comp) package, SBML can instantiate models to construct more complex models, such as the full genetic toggle switch design shown in Fig. 1(c). The dashed arcs in the top-level model of the genetic toggle switch model represents complex formation.

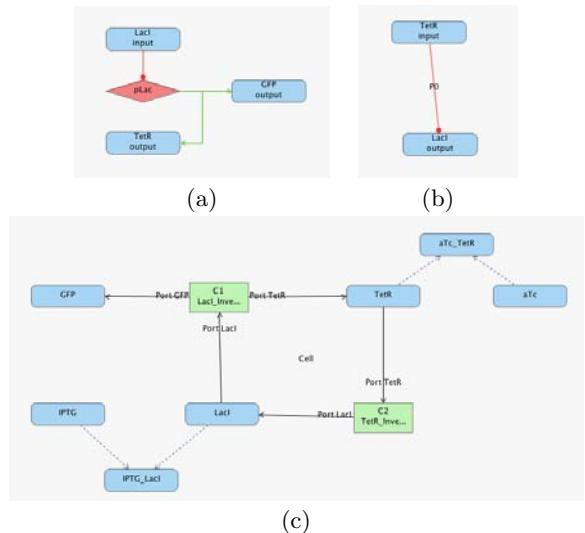


Figure 1: (a) A LacI inverter, (b) a TetR inverter, and (c) a genetic toggle switch in *iBioSim*.

SBOL 2.0 includes *Component Definitions* to describe DNA, RNA, protein, and other types of components. These components can have *Sequences* associated with them, and they can be related to each other through *Interactions*. The Component Definitions and Interactions can be grouped into *Module Definitions*. Module Definitions can be composed hierarchically to form more complex modules. Finally, Module Definitions can use *Model* objects to reference external models written in, for example, SBML.

The conversion begins with an empty *SBOL Document*. Beginning with the top level SBML model, the conversion process recursively converts each sub-model and adds the corresponding data to the SBOL document. This process builds a SBOL Module Definition for each sub-model with a SBOL Model element referencing its SBML model.

Next, each species is converted into a SBOL Component Definition and given a type of *DNA*, *protein*, *small molecule*, etc. If the species has been annotated with sequence information [3], then this can be referred to, as well. For example, species LacI, TetR, and GFP in Fig. 1(a) are converted to Component Definitions of type protein with role transcription factor, and pLac is converted to a Component Definition of type DNA with role promoter.

A *Functional Component* is created within the Module Definition for each species used in the sub-model, and its definition references the corresponding Component Definition for the species. The Functional Component is also marked as being an input, output, or none, and if it is an input or output, it is given a public access type (private, otherwise). For example, within the LacI inverter Module Definition, a public input Functional Component is created for the LacI species that is an instance of the LacI Component Definition.

Next, the converter checks the type of each SBML reaction. A reaction can be an ordinary chemical reaction, a degradation reaction, complex formation reaction, or a genetic production reaction. Each reaction is converted into one or more SBOL Interactions between the corresponding Functional Components. An Interaction in SBOL 2.0 is used to describe the functional relationship between the *reactants*, *products*, and *modifiers* of the reactions. For example, for an ordinary chemical reaction, an Interaction is created that includes all of them as Participants. A degradation reaction includes the degraded protein as a Participant. The complex formation reaction results in an Interaction that includes the separate proteins as reactants and the complex as a product. Finally, the genetic production reaction is converted into several Interactions. In particular, it creates one Interaction for each activator or inhibitor and the promoter, and it creates one production Interaction for each promoter with its products. For example, in Fig. 1(a) the repression arc is converted into an Interaction where LacI is an inhibitor Participant and pLac is a promoter Participant.

In hierarchical SBML models, the same species may appear at various levels of the hierarchy, which is indicated using replacements and replacedBy elements. In particular, a species in the top-level model may have a replacement that states that all instances of the species in the sub-model should be replaced by this top-level species. On the other hand, a replacedBy object indicates a species in the top-level model should be replaced by a species in a sub-model. In SBOL, this operation is accomplished using MapsTo objects. Namely, a MapsTo is used to identify when Component Instances used at different levels of the hierarchy are actually the same Component Instance. A SBOL MapsTo object is created for each SBML replacement or replacedBy object. The MapsTo object maps a local Component Instance to a remote Component Instance. In this case, the local reference is to a Functional Component for the species in the top-level model and the remote is the Functional Component for the species in the sub-model. For a replacement, the MapsTo object has a refinement type of *use local* indicating that the properties of this object should be taken from the Functional Component in the top-level object. For a replacedBy, the MapsTo object has a refinement type of *use remote* indicating that the properties of this object should be taken from the referenced object. For example, in Fig. 1(c), LacI in the TetR_Inverter replaces the top-level LacI which in turn replaces the LacI in the LacI_Inverter. In this case, two SBOL MapsTo objects are created. A visual representation of the generated SBOL is shown in Fig. 2.

3. DISCUSSION

Standards are an important feature of synthetic biology to help overcome the challenges to reproduce designs and reuse

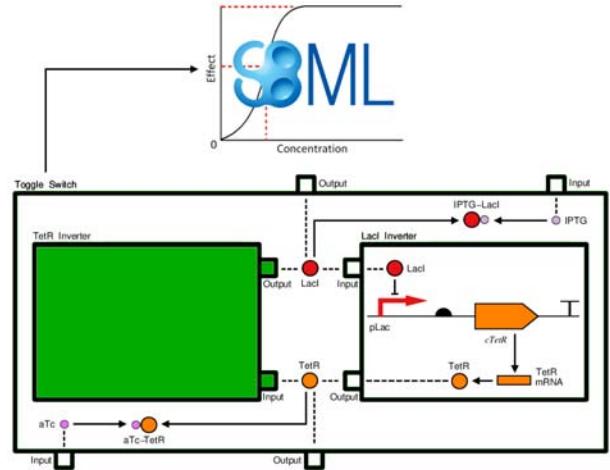


Figure 2: Genetic toggle switch in SBOL 2.0.

published work. While SBML is used to create models for simulation, SBOL is used for the structural design of genetic circuits. This abstract describes a conversion method from SBML to SBOL that extracts the structural and qualitative behavioral information. As a future goal, we plan to attach quantitative information, such as reaction rate constants, species initial amounts, stoichiometry, etc. enabling the ability to round-trip the conversion from SBOL back to SBML. In SBOL, this quantitative data will be represented in SBML using Generic Top Level objects and Annotations.

Acknowledgements

We thank Nicholas Roehner of Boston University for his help with this converter. This material is based upon work supported by the National Science Foundation under Grant Number DBI-1356041. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

4. REFERENCES

- [1] M. Courtot et al. Controlled vocabularies and semantics in systems biology. *Molecular systems biology*, 7(1), 2011.
- [2] M. Hucka et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, Mar. 2003.
- [3] N. Roehner and C. Myers. A methodology to annotate systems biology markup language models with the synthetic biology open language. *ACS Synthetic Biology*, 5(2), 2013.
- [4] N. Roehner, E. Oberortner, M. Pocock, J. Beal, K. Clancy, C. Madsen, G. Misirli, A. Wipat, H. Sauro, and C. J. Myers. Proposed data model for the next version of the synthetic biology open language. *ACS Synthetic Biology*, 4(1):57–71, 2015. PMID: 24896221.
- [5] N. Roehner, Z. Zhang, T. Nguyen, and C. J. Myers. Generating systems biology markup language models from the synthetic biology open language. *ACS Synthetic Biology*, 2015.

SBOL Stack: The One-stop-shop for Storing and Publishing Synthetic Biology Designs

Curtis Madsen

School of Computing Science
Newcastle University, UK
curtis.madsen@ncl.ac.uk

Goksel Misirli

School of Computing Science
Newcastle University, UK
goksel.misirli@ncl.ac.uk

Matthew Pocock

Turing Ate My Hamster LTD
Newcastle upon Tyne, UK
turingatemymhamster@gmail.com

Jennifer Hallinan^{*}

School of Computing Science
Newcastle University, UK
jennifer.hallinan@mq.edu.au

Anil Wipat[†]

School of Computing Science
Newcastle University, UK
anil.wipat@ncl.ac.uk

ABSTRACT

We have developed the SBOL Stack, a Sesame RDF database specifically designed for storing and publishing of SBOL data. The SBOL Stack can be used to publish a library of synthetic parts and designs as a service, to share SBOL with collaborators, and to store designs of biological systems locally. The system includes a Web client that allows users to upload new biological data to the database, a simplified search option that automatically creates SPARQL queries to access desired SBOL parts, and a visualiser for viewing results in a graphical representation. Users of the SBOL Stack can register different instances of the SBOL Stack with their own instance and perform federated queries over all registered databases. Federated queries allow the SBOL Stack to perform automatic data integration. In this paper, we demonstrate how the SBOL Stack is a valuable tool for researchers working on the design of systems in synthetic biology.

1. INTRODUCTION

Synthetic biology is a growing field that combines ideas from biology and engineering with the goal of designing and building new useful biological systems. It is, however, often difficult to utilise the extensive amount of biological data for design in synthetic biology. This difficulty arises because efforts are usually carried out in individual laboratories and carried out by teams in different geographic locations. Moreover, the interests of researchers can vary greatly, and biological data relevant to the design of genetic circuits is typically not exchanged.

Standards are necessary to aid in the interpretation and exchange of biological information. An emerging standard in synthetic biology is the *Synthetic Biology Open Language* (SBOL) [2], a data exchange standard for descriptions of genetic parts, devices, modules, and systems. The goals of this standard are to allow researchers to exchange designs of biological parts and systems, to send and receive genetic designs to and from biofabrication centers, to facilitate storage of genetic designs in repositories, and to embed genetic designs in publications.

Most existing repositories for SBOL simply store individual

^{*}Currently at Macquarie University.

[†]To whom correspondence should be addressed.

SBOL files. However, the SBOL language can be represented in RDF/XML and can thus be stored in triplestore repositories where it can be searched using SPARQL queries similar to the *Knowledgebase of Standard Biological Parts* (SBPkb) [1]. SBOL in the form of RDF automatically integrates into Semantic Web technologies allowing linkage to other biological data, including SBOL in different databases.

We have developed the SBOL Stack to allow researchers to better store, retrieve, exchange, and publish SBOL data. The SBOL Stack is a Sesame *Resource Description Framework* (RDF) [4] database specifically designed for publishing a library of synthetic parts and designs as a service, for storing designs of biological systems locally, and for facilitating the sharing and integration of SBOL with collaborators using Semantic Web technologies. Unlike previously developed repositories like the SBPkb, the SBOL Stack also automatically integrates SBOL data with other RDF data and allows users to perform federated queries over several repositories at once. Additionally, the system includes a Web client that enables the uploading, downloading, and visualisation of SBOL data using SPARQL queries.

2. SBOL STACK

SBOL data, described in XML, can be uploaded to the SBOL Stack through the Web interface. Queries can be performed to retrieve and download SBOL data using either the Web interface or the SPARQL endpoint. These queries allow users of the SBOL Stack to retrieve only desired parts of the SBOL data.

Graph SPARQL queries produce well-formed RDF triples in the structure: *Subject → Predicate → Object*. Graphs generated using these queries can be downloaded and used in other tools that support RDF data. Since the SBOL language is defined using RDF, these types of queries can also return SBOL data. The SBOL Stack has been optimised to perform these types of queries using a search option that automatically performs graph queries without the need to write SPARQL directly. In order to use this option, users simply need to specify a type of Subject, a type of Predicate, and an optional Object.

For graph queries, the SBOL Stack includes a visualiser that presents the results of the query as a collection of nodes con-

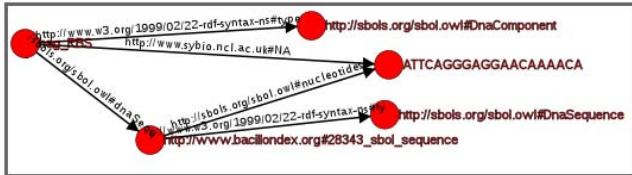


Figure 1: A graphical visualisation resulting from performing a graph query using the SBOL Stack. The directed graph visualisation allows users to explore the data in their repository by performing additional queries on selected nodes.

nected to each other via edges. This visualisation is useful for users as it allows them to see how the data they are interested in is organised in the database. Figure 1 depicts an example of a visualisation that is produced when performing a graph query using the SBOL Stack.

2.1 Data Integration

Information about genetic features, their biological role, and their functional interactions is usually spread over many databases. This situation makes it difficult to automatically assimilate the information necessary for biological system design. Since SBOL is based on RDF, it is ideal for data integration and can easily be linked to other RDF data. Some examples include integrating with ontologies such as the Sequence Ontology and the Gene Ontology.

In addition to SBOL, the SBOL Stack includes semantically-enriched integrated data from the BacillOndex [5] dataset, an ontology about genetic features, gene products and their annotations, gene regulatory networks, metabolic pathways, and so on. It is possible to include other custom ontologies in the SBOL Stack as long as they can be expressed in RDF. Biological entities can be mapped to SBOL objects using the ontology to enrich the data, and the data model from the ontology can be used to automate the identification of biological parts via SPARQL queries.

2.2 Federated Querying

To facilitate exchange, instances of the SBOL Stack can be installed by researchers at various organisations. One of the strengths of the SBOL Stack is its ability to register many databases and perform *federated queries* [3]. Federated queries allow for the retrieval and compilation of more complete data by automatically querying all registered databases without the need to manually query each individual repository. In fact, the SBOL Stack can register any Sesame RDF database, so other repositories that contain information about biological parts can be included in the federated queries. Figure 2 presents a diagram of how federated querying works in the SBOL Stack.

3. DISCUSSION & FUTURE WORK

As more biological data are generated, it will become essential to adopt standards and repositories so computer applications can communicate and exchange data efficiently and in an automated manner. Tools and repositories that support standards like SBOL will be required to create workflows for design in synthetic biology. The automatic retrieval and integration of SBOL data provided by the SBOL Stack makes it a valuable tool for synthetic biology workflows.

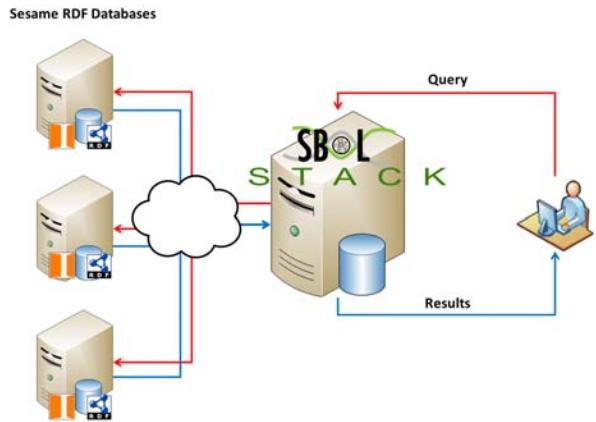


Figure 2: An example of how the SBOL Stack performs federated queries. Here, a user sends a query to an instance of the SBOL Stack which subsequently queries other RDF databases that it has registered. The SBOL Stack then combines the results of all the queries and returns a collection of RDF objects to the user.

The SBOL Stack, in a similar fashion to NOSQL databases, allows for storing data flexibly. As a triplestore, it has the advantage of utilising already available standard libraries and the SPARQL querying language for data retrieval. We are currently developing an API to allow computational tools to programmatically access the SBOL Stack using the direct search interface that is utilised by the Web client. This API will eliminate the need for computational tools to require users to write raw SPARQL queries when accessing the SBOL Stack. Additionally, when performing federated queries, all registered repositories are world-readable but are only writable by registered users. We currently have plans to implement a more sophisticated security system allowing users to give specific read and write permissions to the repositories they create and register. In the future, we also plan to support RDF/JSON output for more lightweight access to data.

4. AVAILABILITY

The SBOL Stack source code is freely available for download under the Apache License, Version 2.0 at <https://bitbucket.org/ncl-intbio/sbolstack>. Our SBOL Stack server can be accessed via www.sbolstack.org.

5. REFERENCES

- [1] M. Galdzicki et al. Standard biological parts knowledgebase. *PLoS ONE*, 6(2):e17005, 02 2011.
- [2] M. Galdzicki et al. Synthetic Biology Open Language (SBOL) Version 1.1.0, BBF RFC 87, 2012.
- [3] P. Jacsó. Thoughts About Federated Searching. *Information Today*, 21(9):17–20, 2004.
- [4] F. Manola and E. Miller, editors. *RDF Primer*. W3C Recommendation. World Wide Web Consortium, February 2004.
- [5] G. Misirli et al. BacillOndex: An integrated data resource for systems and synthetic biology. *Journal of Integrative Bioinformatics*, 10(2):224, 2013.

Stoichiometrically Minimal Source Pathways via Model Checking

Matthew Fong
EECS, UC Berkeley
mfong92@berkeley.edu

Sanjit A. Seshia
EECS, UC Berkeley
sseshia@eecs.berkeley.edu

ABSTRACT

We formulate the problem of finding stoichiometrically minimal source pathways (SMSPs) in biochemical metabolic graphs and present a model checking approach to solve it. SMSPs are paths whose source nodes correspond to native metabolites and which use a non-dominated amount of those compounds. Our approach allows one to eliminate inefficient pathways when selecting the best path to a target. We also investigate the impact of the choice of model checking technique on the runtime for our procedure.

Categories and Subject Descriptors

J.3 [Computer Applications]: Life and Medical Sciences—*Biology and genetics*; B.7.2 [Design Aids]: Verification

General Terms

Algorithms, Design, Verification

1. INTRODUCTION

Enzymatic pathway synthesizers (e.g., [7]) are capable of constructing pathways to target chemicals based on naturally-known reactions as well as reactions that are inferred as plausible. However, few have the capability to provide information on which pathways are better than others, in terms of success likelihood and efficiency. In our work, we make strides towards this goal by defining the problem of finding stoichiometrically minimal source pathways (SMSPs). These SMSPs are defined by their usage of native metabolites, which are compounds biosynthetically accessible from raw sources (e.g., glucose, ammonia, sulfate, and phosphate) using only the enzymes genetically encoded within the host organism. Informally, SMSPs are paths whose source nodes correspond to native metabolites and which use a *non-dominated* amount of those compounds. The SMSP problem is related to that of balancing a metabolic pathway: the latter is a limiting case of the SMSP problem where the coefficients for all cofactors are zero. For many metabolic targets, there are no such balanced paths, but there may be routes that use significantly fewer native metabolites than others.

Problem Statement: We are given a metabolic graph $G = (V, S, U, t)$, where V is the set of all chemicals, $U \subseteq V$ is the set of native metabolites, t is the target, and S is the matrix expressing the coefficients of the reactions, where S_{ji} is the coefficient for chemical i in reaction j . Let $|V| = n$ and $|U| = k$. Cost vector $c = [c_1, \dots, c_k, \dots, c_n]$ represents the amount of each chemical that is used (positive) or produced (negative) in any path. For each hyperedge j that is traversed, decrement c_i by S_{ji} . The solution (SMSPs) for a given G is the set of all non-dominated paths from U to t where the cost of a path is represented by the vector $c = [c_1, \dots, c_k, 0, \dots, 0]$. As this graph is completely connected, there is some ordering of reactions that force entries of the cost vector to

be zero for the non-native metabolites, as we are interested in the stoichiometric cost in terms of only the native metabolites. Path x dominates path y if $c_i^{(x)} \leq c_i^{(y)} \forall i \in U$, and $c_j^{(x)} < c_j^{(y)}$ for at least one j , where $c_i^{(x)}$ and $c_i^{(y)}$ are the costs for the i th chemical in paths x and y , respectively. An example of this formalization can be seen in Section 2.

Finding SMSPs can easily be shown to be NP-complete [4] with a reduction from the set partition problem [5], so we take a model checking approach to solve this SMSP problem. In this work, we describe the model checking approach that we employ, as well as results from an *E. coli* system, provided by the Act Ontology pathway synthesizer [7].

2. MODEL CHECKING FORMULATION

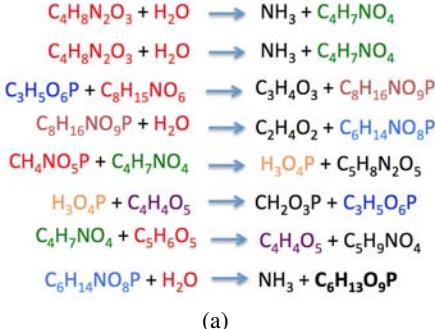
For model checking, the state of the system is defined by a vector $q = [N_1, \dots, N_n, rxn]$, where N_i represents the number of units of compound i , and rxn represents the reaction in the metabolic graph that was most recently traversed (“fired”). The edges of the metabolic graph define the transition function. The initial state, q_0 , is $[0, \dots, 0]$, where the last 0 denotes that no previous reaction was traversed. The target chemical compound defines the target state that must be reached within a finite number of steps. There are limits on each of the N variables, where M is set to be the maximum number of units of a compound (and should be large enough so that it is never reached). rxn can take on the value of any reaction, and can transition to any other valid reaction whose reactants are available. Our approach is illustrated with an example below.

Chemical coefficients (N_1, \dots, N_n) increase when a reaction is fired that produces the chemical, and decrease when a reaction uses the chemical. Below, we define S , U , and t for a simple example:

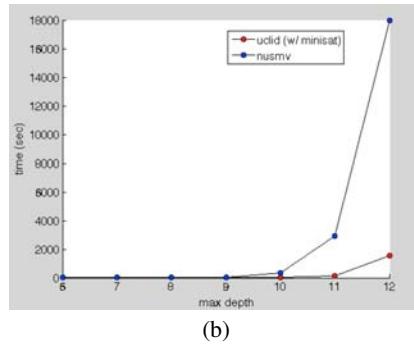
$$2A + B \rightarrow C \text{ (rxn1)} \quad B + 2C \rightarrow D \text{ (rxn2)}$$
$$U = \{A, B\}, \quad t = \{D\}, \quad S = \begin{bmatrix} -2 & -1 & 1 & 0 \\ 0 & -1 & -2 & 1 \end{bmatrix}$$

The number of units of A , B , C , D are denoted by N_1 , N_2 , N_3 , N_4 , respectively. The corresponding transition function for N_3 is given below in the notation of the NuSMV model checker [3]:

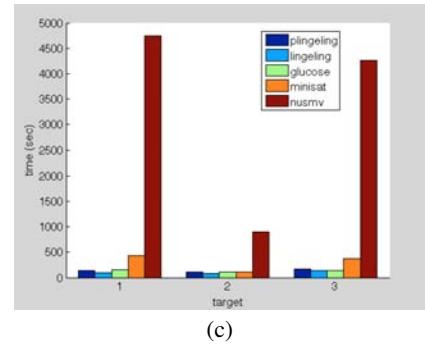
```
next (N3) :=  
case  
  N3 > M : N3;           //stop at upper bound  
  N3 < -M : N3;          //stop at lower bound  
  N4 > 0 : N3;           //stop at target  
  rxn = 1 : N3 + 1;       //transition for rxn1  
  rxn = 2 : N3 - 2;       //transition for rxn2  
  TRUE : N3               //else, is the same  
esac;
```



(a)



(b)



(c)

Figure 1: (a) Sample pathway to $C_6H_{13}O_9P$. Red compounds are native. (b) Runtime vs. max depth for finding paths for $C_6H_{13}O_9P$. (c) Comparison of SAT solvers after encoding with UCLID, depth 11 (targets in (c)): (1) $C_5H_{11}O_7P$, (2) $C_7H_5NO_4$, (3) $C_3H_7O_6P$.

An appropriate linear temporal logic (LTL) [6] specification must then be checked to generate a counterexample that provides one possible path. We iteratively add a constraint to the LTL formula to eliminate this generated counterexample and generate a new counterexample that is not dominated by this original one (or it returns “no counterexample found”). The characteristics that the counterexample must have are: (1) positive coefficient for target, (2) non-negative coefficients for all non-source chemicals, and (3) not dominated by any previously generated counterexample.

Here is an example on another small graph, with target N51. This first LTL specification has characteristics (1) and (2), and is the initial specification that is evaluated.

$$G \ (N51 = 0 \mid (N51 = 0 \mid (N7 < 0) \mid (N20 < 0) \mid (N28 < 0)))$$

The following counterexample is found: $N3 = -1$, $N5 = -1$, $N51 = 1$, implying that one unit each of Chemicals 3 and 5 can be consumed to produce one unit of 51. The next specification is then:

$$G \ (N51 = 0 \mid (N51 = 0 \mid (N7 < 0) \mid (N20 < 0) \mid (N28 < 0)) \mid (N3 \leq -1 \& N5 \leq -1))$$

This continues until no further counterexamples are found.

3. RESULTS

First, we examine a concrete solution to this problem. We are using a metabolic graph for *E. coli* generated by Act [7], which has a total of 1253 reactions and 762 chemicals. In Figure 1(a), we see an example of one non-dominated pathway to d-allose-6-phosphate. Although this example only has compounds with coefficient 1, our system accounts for non-unitary coefficients as well. In this case, the cost of this pathway would be the total of the compounds in red on the reactants side. All other reactants in this pathway come from the products of some other reaction in the pathway.

We begin by encoding our model with NuSMV [3], a symbolic model checker. There are two important points to note in this analysis. First, we must impose a maximum search depth for our model checker. In practice, the most interesting (and best) paths occur within a relatively low maximum depth, slightly greater than the depth of the target to account for pathways that are not completely in series. The main tradeoff that we work with is the exponential nature of runtime vs. search depth, as we can see in Figure 1(b). In addition, we report the amount of time that it takes for a given model to reach a result of “no counterexample found”, as that signifies the end of the search of the entire state space.

With the slow performance of NuSMV at bounds greater than 10, we attempted to use other model checking techniques to solve this problem. In particular, we used UCLID [1, 2], a model checker based on satisfiability modulo theories (SMT) solving. As shown in Figure 1(b), UCLID dramatically improved the runtime. Moreover, UCLID can be used with any back-end Boolean satisfiability (SAT) solver, and varying the SAT solver paired with UCLID yields further improvements in runtime as shown in Figure 1(c).

In this *E. coli* pathway, for the targets beyond depth 3, there was an average speedup factor of 25x, with a maximum of 60x and a minimum of 8x when comparing NuSMV with the standard UCLID solver. We have run this procedure on 19 different targets with UCLID, and for a bound of 11, the average runtime for these is 161 seconds, with a minimum of 11 seconds. Some examples of other chemical targets we analyzed are d-glucosamine-6-phosphate and 3-hydroxypropionaldehyde, which is a component of the antimicrobial compound Reuterin. We are further investigating the quality of pathways past a certain reaction threshold, as well as other methods to speed up our runtime. More details are available in [4].

4. CONCLUSIONS

We have defined the SMSP problem and shown a viable method of finding SMSPs, with the ability to generate all possible SMSPs (bounded by an input search depth). From our experiments, we already see that the choice of model checker can have a large impact on the runtime, which indicates that further optimization could make greater depths more tractable. Future work can address the variance among the average runtimes for different targets, as well as apply similar methods to other optimal path problems in synthetic biology [4].

Acknowledgment: We are grateful to Chris Anderson and Saurabh Srivastava for their advice and assistance throughout this project.

5. REFERENCES

- [1] UCLID Verification System. Available at <http://uclid.eecs.berkeley.edu>.
- [2] R. E. Bryant, S. K. Lahiri, and S. A. Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In *CAV*, LNCS 2404, pages 78–92, July 2002.
- [3] A. Cimatti, E. M. Clarke, E. Giunchiglia, and et al. Nusmv 2: An opensource tool for symbolic model checking. In *CAV*, pages 359–364, 2002.
- [4] M. Fong. Two optimal path problems in synthetic biology. Master’s thesis, EECS Department, University of California, Berkeley, May 2015.
- [5] N. Karmarkar and R. M. Karp. The differencing method of set partitioning. Technical report, Berkeley, CA, USA, 1983.
- [6] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, 1977.
- [7] S. Srivastava, J. Kotker, S. Hamilton, P. Ruan, J. Tsui, J. C. Anderson, R. Bodik, and S. A. Seshia. Pathway synthesis using the Act ontology. In *Proceedings of the 4th International Workshop on Bio-Design Automation (IWBDIA)*, June 2012.

Double Dutch: A Tool for Designing Libraries of Variant Metabolic Pathways

Nicholas Roehner
Boston University, US
nroehner@bu.edu

Douglas Densmore
Boston University, US
dougd@bu.edu

1. INTRODUCTION

The development of technologies for designing novel DNA components [2, 1] has enabled the design of large combinatorial libraries of variant metabolic pathways and genetic circuits. Since it can be difficult to physically construct and screen these libraries in their entirety, new tools are needed to design these libraries for efficient testing. To meet this need, we have developed Double Dutch, a web application that tailors libraries of variant pathways for use in a design of experiments (DOE) framework.

In the context of synthetic biology, DOE techniques can be used to restrict testing to only those variants that are statistically relevant for determining the relationship between variant parameters and measured performance. Despite this potential, there are few published instances of applying DOE methods to synthetic biology [4]. Currently, there exist general purpose DOE software tools, such as JMP [3], that service the biological sciences with varying degrees of specificity, but none have been explicitly developed for synthetic biology. To bridge the gap between biological and experimental design, Double Dutch automates the process of mapping from the coding sequences (CDS) and other characterized DNA components that make up variant pathways to the factors and levels that define the conditions of a full factorial experiment. The end result is a library of variant pathways that can be used in a DOE framework. Figure 1 presents an overview of this mapping process.

2. GRAMMAR

In order to determine which DNA components are eligible for mapping to the factors and levels of an experimental design, Double Dutch implements a formal grammar. The rules of this grammar specify that experimental factors must be implemented as partial genes that include at least one CDS, while the levels that each factor takes on must be implemented as parameterized DNA components that regulate gene expression, such as promoters, ribosome binding sites (RBS), and terminators. While the examples in this abstract focus on mapping RBS-CDS pairs to factors and mapping promoter-terminator pairs with REU measures of their transcription strengths to levels, Double Dutch is capable of supporting other use cases through its grammar. These include mapping promoter-CDS-terminator combinations to factors and mapping RBSs with REU measures of their translation strengths to levels, or mapping promoter-RBS-CDS combinations to factors and mapping terminators with measures of their relative efficiencies to levels.

3. LEVEL ASSIGNMENT

Once uploaded DNA components and parameters are classified as candidate factors or levels via a grammar, a Double Dutch user only needs to select the partial gene factors in their pathway of interest and choose a desired number of levels per factor. Double Dutch then uses heuristic algorithms, most notably k-means clustering and simulated annealing, to automate the process of assigning candidate DNA components to the levels of the experimental design. Prior to assignment, all candidate components are partitioned into k clusters based on their parameter values, where k is the chosen number of levels per factor. The mean parameter values of these clusters set the target values for each level, while the clusters themselves filter the candidates available for assignment to each level. Double Dutch also allows users to manually set these target values if desired.

Level assignments are costed as the weighted sum of three concerns: level matching, pathway homology, and component reuse. Double Dutch attempts to manage these conflicting concerns according to user-defined weights and find the level assignment with the smallest cost by randomly changing which DNA components are assigned to each experimental level. Each change is accepted or rejected in accordance with a simulated annealing heuristic. Under this heuristic, changes that increase the cost by a large amount are more likely to be accepted early on, which can help prevent entrapment in a local minimum.

In the case of level matching, Double Dutch attempts to minimize the quantitative differences between the parameters of the assigned DNA components and the target values of the experimental levels. In the case of pathway homology, Double Dutch attempts to minimize the number of homologous DNA components within each variant pathway of the resultant library, so as to reduce the risk of homologous recombination during pathway construction. Finally, in the case of component reuse, Double Dutch attempts to maximize the reuse of DNA components across variant pathways and thereby minimize the costs associated with modular cloning.

4. PRELIMINARY RESULTS

As a demonstration of Double Dutch's level assignment capability, Figure 2 shows the results of designing pathway libraries for experiments containing five to nine factors and two to five levels. In particular, the top of Figure 2 displays the costs of the best level assignments found by Double Dutch after 500 trials, while the bottom compares these assignments with the best found by a purely random approach. In this example, all three assignment concerns are

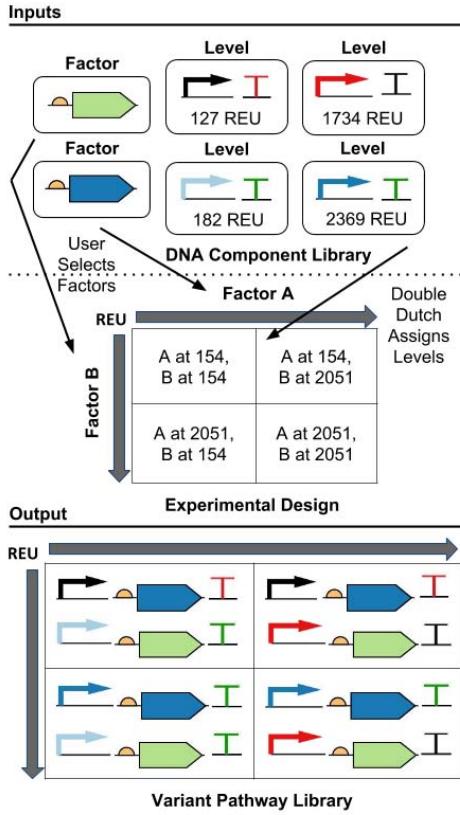


Figure 1: Overview of Double Dutch library design. DNA components are assigned to the factors and levels of a full factorial experimental design to produce a library of variant metabolic pathways.

weighted equally. In addition, all DNA components belong to a library containing 1,069 promoter-terminator pairs that have been characterized for transcription strength in yeast.

As shown in Figure 2, the cost of the level assignment found by Double Dutch generally increases as the size of the experimental design increases. In addition, the percentage by which Double Dutch outperforms random assignment generally decreases, though not to less than 25 percent for the largest designs. One cause of these effects is that, as the size of the experimental design approaches the limit of what the DNA component library can implement without introducing pathway homology, the level matching and component reuse costs are outweighed by a large pathway homology cost that dominates the total. Finally, the time taken by Double Dutch to perform 500 trials of level assignment increases with design size, but it scales tolerably and is less than six minutes for the largest designs in this example.

5. CONCLUSIONS

Double Dutch is among the first software tools capable of designing combinatorial libraries of variant metabolic pathways that are tailored for use in a DOE framework. While this framework relies on generic statistics software to prune variants for testing and fit the resulting data to an empirical model, we are currently implementing the same techniques in Double Dutch and seeking to customize them for use in

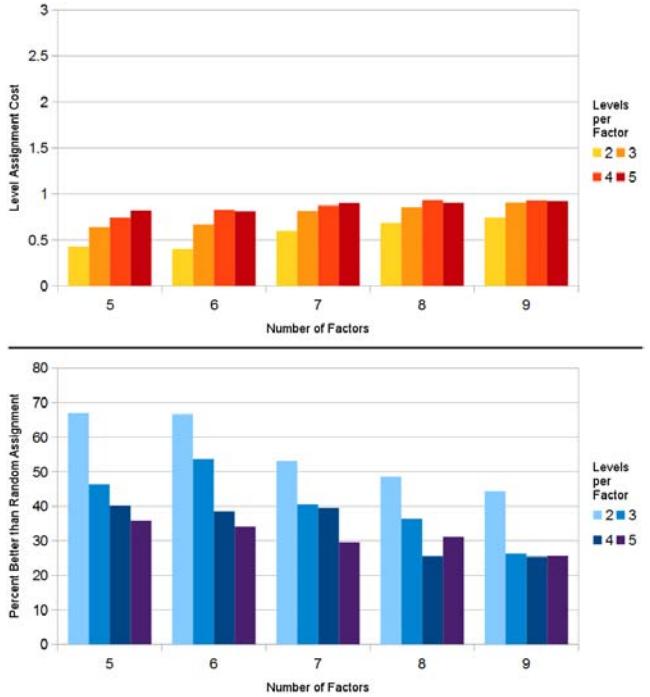


Figure 2: Costs of best level assignments found by Double Dutch (top) and the percentages by which Double Dutch outperforms random level assignment (bottom). The worst possible cost is three.

a synthetic biological context. Ultimately, Double Dutch can be used to design libraries that provide better coverage of pathway design spaces, minimize the risk of homologous recombination, and reduce the monetary cost of modular cloning. Double Dutch is currently closed source, but the application can be accessed at www.doubledutchcad.org.

6. ACKNOWLEDGMENTS

We thank Benjamin Gordon, Eric Young, and the other members of the MIT-Broad Foundry for their aid in conceiving this project and providing characterization data for testing purposes.

7. REFERENCES

- [1] Y.-J. Chen, P. Liu, A. A. Nielsen, J. A. Brophy, K. Clancy, T. Peterson, and C. A. Voigt. Characterization of 582 natural and synthetic terminators and quantification of their design constraints. *Nature Methods*, 10:659–664, 2013.
- [2] H. M. Salis, E. A. Mirsky, and C. A. Voigt. Automated design of synthetic ribosome binding sites to control protein expression. *Nature Biotechnology*, 27:946–950, 2009.
- [3] J. Sall, A. Lehman, M. L. Stephens, and L. Creighton. *JMP start statistics: a guide to statistics and data analysis using JMP*. SAS Institute, 2012.
- [4] M. Welch, S. Govindarajan, J. E. Ness, A. Villalobos, A. Gurney, J. Minshull, and C. Gustafsson. Design parameters to control synthetic gene expression in *Escherichia coli*. *PLoS ONE*, 4, 2009.

Millstone: Software for iterative genome engineering

Gleb Kuznetsov^{*}
Biophysics Program
Harvard University
Boston, MA
kuznetsov@g.harvard.edu

Daniel B. Goodman^{*}
Harvard-MIT Health Sciences
and Technology / BIG
Cambridge, MA
dbg@mit.edu

Marc J. Lajoie^{*}
Department of Biochemistry
University of Washington
Seattle, WA
mlajoie@uw.edu

George M. Church
Department of Genetics
Harvard Medical School
Boston, MA
gchurch@genetics.med.harvard.edu

ABSTRACT

The price of sequencing has fallen below \$20 per microbial genome [1], and advances in genome editing allow for the generation of billions of combinatorial genomic variants per day [2]. Computational analysis at this scale is a rate-limiting step in microbial genome engineering. We describe *Millstone*, a web-based platform for iterative genome engineering and multiplex mutation analysis. *Millstone* can handle the complexity of re-sequencing, variant-calling, and genotype comparison for hundreds of microbial genomes, as well as the design of targeted mutations in successive rounds of experiments. We describe how we used *Millstone* as a guide to improve the fitness of an engineered strain of *E. coli*. *Millstone* is open source and available as an Amazon Machine Image (AMI), making it a scalable solution accessible to any lab.

1. INTRODUCTION

Microbes possess a staggering amount of genomic diversity, enabling them to evolve and adapt to diverse environments. Sequencing populations of genomes allows us to study this diversity and identify novel phenotypes. In addition to studying natural evolution, biologists can generate targeted genomic diversity in a population of cells using techniques like MAGE [2]. These populations can then be screened or selected for phenotypes which are useful for biotechnology or for answering basic biological questions. Multiplex sample preparation allows microbes such as *E. coli* to be sequenced in bulk for under \$20 per sample at 30x coverage [1].

The ease of generating and sequencing evolved and rationally modified populations of bacteria lends itself to an iterative cycle of combinatorially introducing targeted mutations and converging on phenotypes of interest. At each iteration, whole genome sequencing data and assay measurements must be converted into actionable designs for follow-up experiments. The complexity and effort required to convert raw sequencing data for hundreds of genomes into a representation of mutations that allow for querying and comparison can overwhelm in-house computational infras-

tructure and expertise and limit the rate of this engineering cycle. Existing tools allow users to pipeline custom alignment and analysis steps, but are not optimized for large amounts of data and are not capable of comparative analysis among multiple genomes. An ideal solution would integrate features such as interactive querying, visualization, collaboration, iteration, genome versioning, and the design of additional mutations or reversions.

To fill the need for an integrated software solution, we developed *Millstone*, a web-based platform that facilitates an iterative approach to genome engineering. *Millstone* allows non-computational researchers to explore the diversity of their evolved or engineered bacterial strains and to design follow-up experiments. *Millstone* grows out of experience and needs in our own lab, and development versions of the software were used extensively in various projects, including reassigning the UAG codon genome wide in *E. coli* [3] and identifying escape-conferring mutations while engineering a biocontained strain of *E. coli* [4].

2. FEATURES

A typical workflow and major components of *Millstone* are illustrated in Figure 1. A researcher provides a starting reference genome (.fasta or .genbank) and specifies an initial list of target mutations. *Millstone*'s *optimage* integration can generate oligonucleotides for use with MAGE. Following experiments and whole genome sequencing, the researcher uploads raw sequencing reads (.fastq). *Millstone* then performs alignment (*bwa*), single nucleotide variant (SNV) calling (*freebayes*), and structural variant (SV) calling (*lumpy*). *Millstone* then parses the called SVs and SNVs into a unified data model. A custom query language allows searching and filtering efficiently over the data. Genotype calls often need to be checked by eye, and *Millstone*'s variant analysis view provides programmatically-generated links to visualizations of the relevant read alignments in *JBrowse*. *Millstone* also allows genome design iteration and versioning, where new reference genomes can be generated from sets of called and triaged variants. This allows the next round of genomic mutations to be aligned back to the most recent ancestral strain.

*These authors contributed equally to this work.

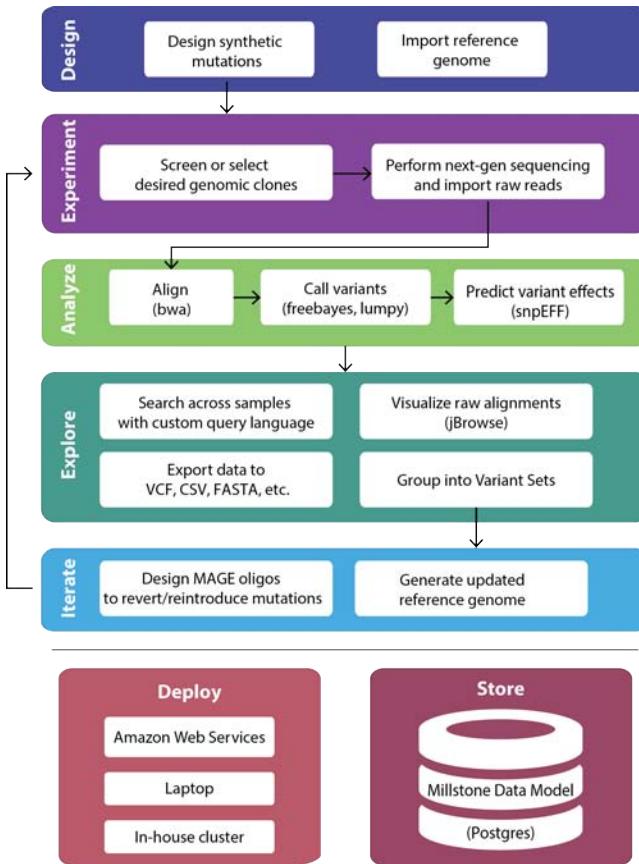


Figure 1: Millstone components and example workflow.

Millstone is implemented as a Django web application backed by a Postgresql database, which lends flexibility in deployment and scaling, and facilitates collaboration. The software can be deployed on a laptop, an in-house cluster, or on Amazon Web Services (AWS). We recommend AWS for most users and maintain a public release of an Amazon Machine Image (AMI) preconfigured (*cloudbiolinux*) with the latest stable version of *Millstone*. The *Millstone* source code is available at <https://github.com/churchlab/millstone>.

3. CASE STUDY

We used *Millstone* to guide experiments that improved the growth rate of the Genomically Recoded Organism (GRO) created in [3]. In that work, a sub-strain of *E. coli*, C321. ΔA , was created where all 321 known instances of the UAG stop codon were replaced with the synonymous UAA codon, and the UAG translational release factor *prfA* was deleted, freeing up the UAG codon for the incorporation of non-standard amino acids [4]. However, in addition to the 321 designed changes, the strain accumulated 355 additional mutations during construction, and we hypothesized that a subset of these off-target mutations impaired fitness.

Millstone was used to perform variant calling and annotation on C321. ΔA strain. Mutations were exported and filtered by gene essentiality and other parameters to arrive at a candidate set of 127 mutations marked for reversion. We then used *Millstone*'s *optmge* [2] feature to generate oligonucleotides that could be used for 50 cycles of MAGE with this set of mutations. After performing MAGE, we

sequenced 96 strains, including reference controls, and once again ran the sequencing data through *Millstone* to call variants. The annotated variants identified by *Millstone*, in combination with additional analysis, allowed us to revert the most likely causal SNPs, creating a strain with an improved growth rate.

4. DISCUSSION

Advances in genome engineering technologies make it possible to rapidly generate microbial populations with a tremendous amount of diversity. This creates an opportunity to interrogate the relationship between genotype and phenotype, but is limited by the availability of analytical tools for efficiently identifying and following up on mutations of interest. *Millstone* enables an iterative approach to genome engineering by providing a single workflow for designing targeting oligonucleotides, calling mutations, and analyzing results.

Users can benefit from all or a subset of *Millstone* features. For example, users who already have raw sequencing data from as many as hundreds of bacterial genomes can use *Millstone* to identify and explore mutations. We have reduced the barrier for other labs to get started with *Millstone* by providing an integration with AWS. Instructions and an online demo are available at <http://churchlab.github.io/millstone>.

Future work on *Millstone* includes extensions to enable CRISPR gRNA design, more sophisticated models of genome versioning, and integration with rule-based genome design tools also under development in our lab.

5. ACKNOWLEDGMENTS

This work was supported by the U.S. Department of Energy (DE-FG02-02ER63445), Defense Advanced Research Projects Agency (HR0011-13-1-0002), AWS in Education Grant award, U.S. Department of Defense National Defense Science and Engineering Graduate Fellowship (G.K., M.J.L.), NSF Graduate Research Fellowship (D.B.G.), and the MIT Undergraduate Research Opportunities Program (K.Y.C., C.C., B.W.A.).

6. ADDITIONAL AUTHORS

Kevin Y. Chen (MIT), Changping Chen (MIT), Michael G. Napolitano (Dept of Genetics, Harvard Medical School), Brian W. Ahern (MIT)

7. REFERENCES

- [1] M. Baym, S. Kryazhimskiy, T. D. Lieberman, H. Chung, M. M. Desai, and R. Kishony, “Inexpensive multiplexed library preparation for megabase-sized genomes,” *bioRxiv*, p. 013771, 2015.
- [2] H. H. Wang, F. J. Isaacs, P. A. Carr, Z. Z. Sun, G. Xu, C. R. Forest, and G. M. Church, “Programming cells by multiplex genome engineering and accelerated evolution,” *Nature*, vol. 460, no. 7257, pp. 894–898, 2009.
- [3] M. J. Lajoie, A. J. Rovner, D. B. Goodman, H.-R. Aerni, A. D. Haimovich, G. Kuznetsov, J. A. Mercer, H. H. Wang, P. A. Carr, J. A. Mosberg, *et al.*, “Genomically recoded organisms expand biological functions,” *Science*, vol. 342, no. 6156, pp. 357–360, 2013.
- [4] D. J. Mandell, M. J. Lajoie, M. T. Mee, R. Takeuchi, G. Kuznetsov, J. E. Norville, C. J. Gregg, B. L. Stoddard, and G. M. Church, “Biocontainment of genetically modified organisms by synthetic protein design,” *Nature*, 2015.

MERLIN: a DNA Design Tool for Large-Scale Genome Engineering

Michael Quintin, Samir Ahmed, Swapnil Bhatia,
Douglas Densmore
Cross-Disciplinary Integration of Design Automation
Research (CIDAR)
Boston University
mquintin@bu.edu

Aaron Lewis, Natalie Ma, Farren Isaacs
Department of Molecular, Cellular, & Developmental
Biology, Systems Biology Institute
Yale University

INTRODUCTION

Here we describe Merlin (<http://merlincad.org>), a web-based tool to assist biologists in designing experiments using Multiplex Automated Genome Engineering (MAGE).

Merlin provides methods to generate the pool of single-stranded DNA oligonucleotides for a MAGE experiment. These oligos are designed not only for optimal recombination efficiency, but also to minimize off-target interactions. The application further assists experiment planning by reporting predicted allelic replacement rates after multiple experiment cycles, and enables rapid result validation by generating primer sequences for Multiplexed Allele-Specific Colony (MASC) PCR [2].

Categories and Subject Descriptors

J.3 [Computer Applications]: Biology and genetics; J.6 [Computer-aided Engineering] Computer-aided design (CAD)

General Terms

Design, Experimentation

Keywords

Bioengineering, genomics, multiplex automated genome engineering, MAGE, synthetic biology

1. BACKGROUND

MAGE utilizes homologous recombination proteins originally isolated in phage to achieve scarless integration of synthetic ssDNA (oligos) into a bacterial genome. The structure of these oligos consists of 5'- and 3'-terminal homology arms that are complementary to the sequence flanking the targeted locus, and a central sequence corresponding to the desired mutation. Once introduced into the cell by electroporation, oligos are proposed to anneal to the lagging strand of the replicating bacterial chromosome with the assistance of phage recombination proteins. The oligos are then integrated into the growing genome in a process mimicking the natural joining of Okazaki fragments on the lagging strand [1].

MAGE can be used to rapidly induce short sequence changes at many targeted loci in a bacterial genome. It is an efficient technique used to construct highly modified organisms, or with a pool of degenerate oligonucleotide sequences to create diverse populations and explore a large genome landscape.

Computer-aided design software plays an important role in synthetic biology. Many seemingly straightforward tasks such as PCR primer design or creating MAGE oligos with the modified bases at a predetermined position can be performed “well enough” by hand, yielding a suboptimal result in order to save time and effort. By incorporating insights from recent technological

advances, Merlin is designed with the intention of increasing the mathematical rigor applied to oligo design without requiring extra effort on the user's part, and to provide a predictive framework to help guide experimental procedures.

2. METHODS

2.1 Oligo Design

The method used to generate oligonucleotides is derived from prior work available as the optMAGE software package [2,4,5]. In addition to the results of its own calculation, Merlin also reports oligos as calculated with the optMAGE script.

The Merlin algorithm for generating an oligo pool given a list of target locations on the genome is as follows:

1. Alter the reference sequence string according to the changes specified by the user.
2. At each target site, isolate every subsequence of nucleotides of the specified oligo length that spans the target.
3. Calculate the free energy of a single-stranded DNA oligo with each of the identified possible sequences.
4. Use BLAST to find the relative likelihood of off-target interactions against the reference genome and any other accepted oligos.
5. Select the subsequence with the best score from these calculations.

This method is considered an improvement on optMAGE by virtue of calculating values for all possible oligos, as well as by the addition of the BLAST step. The Merlin interface reports the free energy and sequence homology results for each optimized oligo along with its optMAGE-calculated counterpart (**Fig. 1**).

2.2 Allelic Replacement Efficiency

MAGE experiments typically consist of multiple cycles transfection. Merlin is capable of creating visualizations based on the calculated probability of each oligo becoming incorporated in the target genome for each cycle. These statistics are useful for predicting how many cycles will be necessary to create a population with a specific diversity of modifications, or how many cycles will be needed to produce an organism which is modified at all target sites.

The allelic replacement efficiency (ARE) over multiple cycles can be modeled as a binomial process in cases where modifications are occurring at discrete sites [4]. The average ARE is determined, then modified by an empirically determined “pooling factor” that accounts for decreased efficiency dependent on the size of the oligo pool.

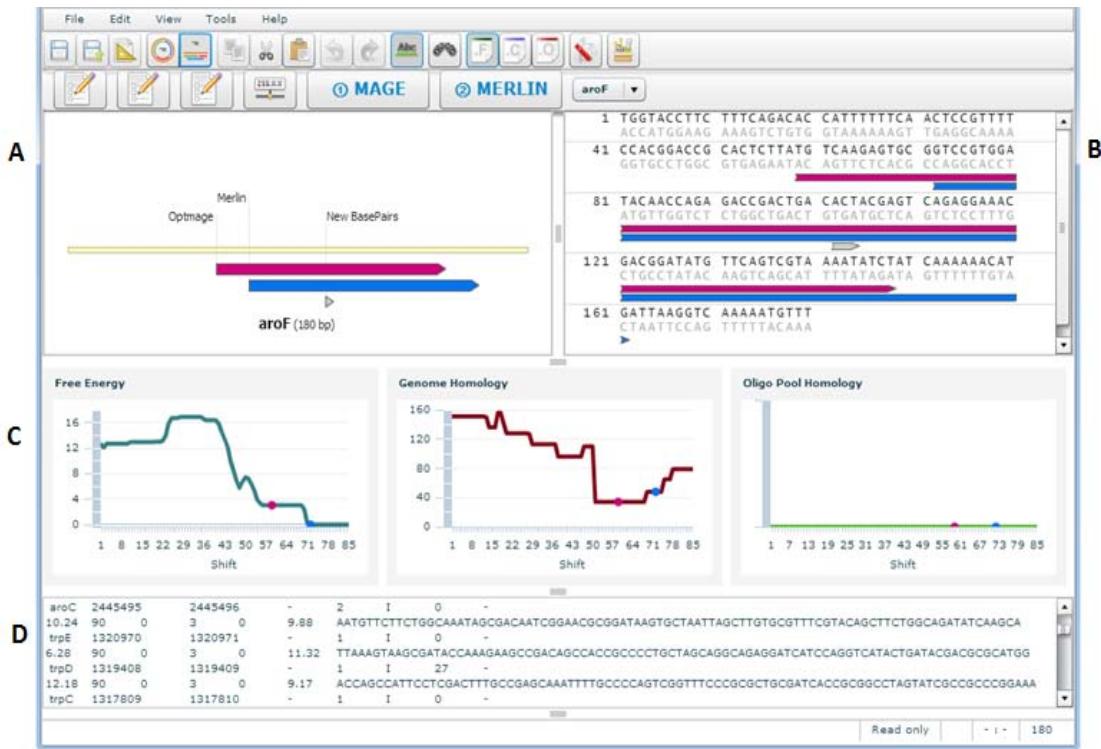


Figure 1. The Merlin interface, containing A) visualization of the location of the modification for one target on the Merlin and optMAGE oligos, B) The aligned oligo sequences, C) (left to right) the free energy, genome BLAST score, and oligo-oligo homology score for all possible oligos covering the current target, and D) text output of oligo prediction results.

2.3 MASC PCR Primer Generation

Multiplexed Allele-Specific Colony PCR allows for simultaneous screening of short mutations at many loci in a single PCR reaction by generating DNA fragments of different sizes for each locus that can easily be distinguished, and is valuable for reducing the amount of manual labor involved in validation [2]. Generating a set of primers for wild-type and mutant genotypes of each target is not straightforward, particularly when screening for short sequence changes. Merlin is capable of generating these primer sets automatically.

2.4 Infrastructure

The interface for Merlin is built using VectorEditor (available at <https://github.com/JBEI/vectoreditor/>), an open source web based DNA sequence analysis and editing tool maintained by the Joint BioEnergy Institute (JBEI).

Free energy and primer melting temperature calculations are performed with the UNAFold software package [3].

Source code for Merlin is available at the CIDAR Github repository (<https://github.com/CIDARLAB/>) under the BSD 3-Clause License (<http://opensource.org/licenses/BSD-3-Clause>).

3. REFERENCES

- [1] Gallagher, R.R., Li, Z., Lewis, A.O., and Isaacs, F.J., 2014. Rapid Editing and Evolution of Bacterial Genomes Using Libraries of Synthetic DNA. *Nature Protocols* 9, 2301-2316.
- [2] Isaacs, F.J., et al, 2011. Precise Manipulation of Chromosomes in Vivo Enables Genome-Wide Codon Replacement. *Science* 333, 348-353.
- [3] Markham, N. R. and Zuker, M. (2008) UNAFold: software for nucleic acid folding and hybridization. In Keith, J. M., editor, *Bioinformatics, Volume II. Structure, Function and Applications*, number 453 in *Methods in Molecular Biology*, chapter 1, pages 3–31. Humana Press, Totowa, NJ. ISBN 978-1-60327-428-9.
- [4] Wang, H.H. and Church, G.M, 2011. Multiplexed genome engineering and genotyping methods applications for synthetic biology and metabolic engineering. *Methods Enzymol.* 498, 409–426.
- [5] Wang, H.H., et al, 2009. Programming cells by multiplex genome engineering and accelerated evolution. *Nature* 460, 894-898.

Software for Engineering Biology in a Multi-Purpose Foundry

Benjie Chen
Ginkgo Bioworks
benjie@ginkgobioworks.com

Barry Canton
Ginkgo Bioworks
barry@ginkgobioworks.com

Dan Cahoon
Ginkgo Bioworks
dpcahoon@ginkgobioworks.com

Austin Che
Ginkgo Bioworks
austin@ginkgobioworks.com

ABSTRACT

Organick is a software that helps user design and execute protocols for building and evaluating biological systems. Organick organizes laboratory procedures as operations in workflows, tracks sample provenance, and manages data.

1. INTRODUCTION

The Bioworks1 foundry at Ginkgo Bioworks aims to make building and evaluating biological systems scalable. Users of Bioworks1 submit requests for PCR, transformation, genome integration, genomic and plasmid DNA preps, protein and metabolite extraction, metabolomic assays, and other services. Foundry operators fulfill requests using liquid handling robots and instruments and return samples and data back to users. Used internally at Ginkgo Bioworks, Bioworks1 lets users design and use standardized processes for 1) building and testing large numbers of microbial organisms, and 2) creating new synthetic biology building blocks such as validated regulatory parts, enzymes, and chassis strains.

Users and foundry operators use a software tool called Organick to plan protocols, track samples, and request foundry services. Organick keeps track of sample contents and locations in containers. Organick lets users construct, using a graphical interface, multi-step asynchronous workflows. Each step in a workflow corresponds to a laboratory procedure (automated or manual, supervised by the user) or a request for foundry service (performed by foundry). Organick also helps users maintain sample provenance when creating new samples, interface with the automation software, and review measurements on samples across multiple steps (e.g. normalizing quantified protein amount by OD readings on original culture sample).

2. SYSTEM DESIGN

2.1 Samples

Sample tracking improves efficiency and throughput of many laboratory processes and follow up analysis efforts. Knowing locations and contents of samples in containers, for example, allows software to automate sample preparation and reaction setup using robots. Recording sample provenance and transfer amounts allows easy association of data from a sample with that sample's parent or child samples, and simplifies data normalization and planning of additional steps based on data.

Organick maintains an inventory of *Samples* and *Containers*. Each sample has an unique ID, a volume, an optional barcode, a location (container ID, row, and column), and some contents. A sample corresponds to either a well in a

non-individually barcoded container (e.g. PCR plate), or an individually barcoded tube. A sample can contain *DNA molecules*, *Strains*, and/or *Reagents*. A strain contains DNA molecules that represent genomic and/or plasmid DNA. A sample containing a DNA molecule represents a DNA sample, whereas a sample containing a strain represents a sample of cells. Organick records content concentrations as well, either obtained from instruments or vendors, or computed after sample manipulations. Keeping concentrations allows Organick to assist in experiment planning. For example, Organick can compute how much to aliquot from a reagent sample for a buffer recipe.

2.2 Operations

Organick represents laboratory procedures as *Operations*. An operation in Organick takes samples and parameters (e.g. volumes to transfer from each sample) as inputs, creates new samples if necessary, updates sample information, and outputs samples.

When Organick executes an operation, it adjusts sample volumes, contents, and concentrations, and, for each liquid transfer from source samples to a target sample, records the time and amount of liquid transferred from each source sample to the target sample. Keeping transfer time and amount allows Organick to normalize data against sample preparation procedures. For example, the concentration of a compound in two samples may measure to be the same, but the two samples may have been diluted differently prior to measurement.

Some example operations are: *Liquid Transfer* aliquots part of a sample to a new sample, *Mix Samples* transfers portions of several samples to a new sample, *PCR* uses Mix Samples to mix template and primer samples, then adds new product molecule to output sample, *Transformation* takes a sample of a strain with a single genomic DNA molecule, a sample of a plasmid DNA molecule, and creates a new sample of a strain with both the genomic and plasmid DNA molecules. If a laboratory procedure does not have a corresponding operation, users can typically use the Mix Samples operation then update sample contents if necessary.

2.3 Service Requests

A *Service Request* in Organick represents a request for a foundry service. It includes service name, input samples, and service specific parameters (e.g. PCR condition).

2.4 Data

Organick represents data from instruments as *Datasets*. Each dataset has a sample, a type, instrument settings and analysis parameters, and data (e.g. raw data from mass

spec). Organick does not fetch data directly from instruments; separate automation programs interface with instruments, fetch data, perform computational analysis, and upload results to Organick. External tools can also create additional analyses based on datasets; these analyses can be uploaded to and displayed on Organick.

Users or external analysis tools can also add *Measurements* to samples. Each measurement has a sample and a predicate, object, value, unit tuple (e.g. ("concentration", compound, 1, "g/L"), or ("OD", nil, 2.1, nil)).

2.5 Workflows

A *workflow* consists of steps. Outputs of a step may become inputs of other steps, thus forming a graph. A step includes operations or service requests. For example, a PCR step may have 9 input samples forming 3 operations – each operation has template, forward and reverse primer samples.

User creates and modifies workflows using a web-based GUI. User can add steps to a workflow at anytime, even after some steps have completed. When adding a step, user specifies the type of operation or service, and their inputs.

Physical completion of a step is decoupled from Organick and performed in the lab; Organick provides user a link and inputs to the automation platform. Upon completion of a step's physical work, user runs the step's operations on Organick to complete sample tracking and the step.

When specifying input samples for an operation or service request, user can select output samples from completed upstream steps, or use “promises” representing future outputs from upstream operations. In the latter case, Organick replaces a promise with real samples when they become available. If an upstream operation produces duplicates (e.g. picked multiple colonies), Organick duplicates the waiting operation, once for each duplicate. Supporting promises allows user to set-up and Organick to validate all steps of a workflow upfront, prior to running any step of the workflow.

Each foundry service has a Organick workflow template. Operators fulfill requests by creating a new workflow from the template and completing steps in the workflow. Organick automatically sets-up operations in each step of the workflow with request input samples and parameters. When a step in an user's workflow includes service requests, Organick looks for completed steps from the service workflow and updates the user's workflow with these steps, showing just output samples.

In each step, Organick displays measurements collected on output samples. Organick can also map measurements from downstream samples onto an upstream step's output samples, so user can review data and add follow up steps from the upstream step.

3. EXPERIENCE

The design of Organick has evolved several times in the past 6 years. Currently it records more than 530K samples and 570K liquid transfers among them.

Screening enzyme sequences for specific substrate activity is a recurring process at Ginkgo Bioworks. Over the past two years, over 500 different protein sequences have been screened under various substrate concentrations. This effort resulted in over 5000 reactions and over 18K datasets. Over 100 workflows capture mixing of protein extract samples with substrate samples, taking reaction timepoints, and sampling timepoints on mass spec. One such workflow in-

cludes 6 steps with 90 Mix Samples operations creating 90 reaction samples per step, 6 steps with 90 Timepoint operations creating 270 timepoint samples per step, and 1 Mass Spec step creating 1634 datasets. User planning for this workflow mostly involves configuring operations in the Mix Samples step so each reaction has the desired concentration and amount of substrates and enzymes.

Integrating DNA into yeast genome is another recurring process. Key steps in a typical workflow include: PCR step to amplify integration cassette from plasmids, Transformation step to integrate the cassette into the genome, and PCR step to amplify the modified region and screen for positives. When planning this workflow, user needs to: a) for operations in the first PCR step, pair template and primer samples, b) for operations in the Transformation step, pair output promises from the PCR step with cell samples, and c) for operations in the last PCR step, pair output promises from the Transformation step with verification primer samples. For each integration event, user sets-up only one operation for the last PCR step, even though multiple colonies from the transformation step may be selected and screened; Organick automatically creates the duplicated operations for the last PCR step, once for each colony.

4. RELATED WORK

Organick is similar to several other systems in literature. Like Puppeteer [3] it lets users plan workflows, but focuses more on sample management and relies on other software for interfacing with instruments. Like Antha [1] it structures workflows as multi-step graphs, but provides a graphical interface rather than a programming interface to graphs. Unlike Diva [2], Organick focuses more on protocol execution and leaves the design of biological systems and protocols to human experts or other software. Additionally, many tools help users analyze data, but few map data onto sample preparation workflows and allow user to further adjust/extend workflow based on data.

5. FUTURE WORK

Ongoing improvements to Organick includes ensuring each input sample to a workflow have enough volume for all operations using the sample, alerting user of plate layout constraints imposed by foundry services or robotic procedures, helping users design plate layouts compatible with these constraints, and *a priori* validation of workflow steps against desired biological system design.

6. ACKNOWLEDGMENTS

Organick is the result of significant contributions by all current and past employees of Ginkgo Bioworks.

7. REFERENCES

- [1] <https://www.antha-lang.org/>. June 2015.
- [2] J. Chen, G. Goyal, H. Plahar, J. Keasling, N. Stawski, and N. Hillson. Diva: More science, less dna construction. *Proceedings of IWBDA 2014*.
- [3] V. Vasilev, C. Liu, T. Haddock, S. Bhatia, A. Adler, F. Yaman, J. Beal, J. Babb, R. Weiss, and D. Densmore. A software stack for specification and robotic execution of protocols for synthetic biological engineering. *Proceedings of IWBDA 2011*.

YeastFabCAM: Computer Assisted Manufacturing for constructing large-scale genetic parts registries

Yisha Luo

Edinburgh Genome Foundry
University of Edinburgh
Edinburgh EH9 3BF, UK

Yue Shen

School of Biological Sciences
University of Edinburgh
Edinburgh EH9 3BF, UK

Emily Scher

School of Biological Sciences
University of Edinburgh
Edinburgh EH9 3BF, UK

Junbiao Dai*

School of Life Sciences
Tsinghua University
Beijing 100084, China

Yizhi Cai*

School of Biological Sciences
University of Edinburgh
Edinburgh EH9 3BF, UK

1. INTRODUCTION

Standardization of genetic parts enables fast assembly of synthetic genotypes, and rigorous functional characterization is the key to rapid prototyping of the desired phenotypes. However, large scale and well characterized eukaryotic parts are still underrepresented as a synthetic biology community resource. Recently, we have launched an international effort to design, construct and characterize around 18,000 genetic parts from the model eukaryote *Saccharomyces cerevisiae* genome (termed YeastFab[1]). These yeast parts were “mined” out of the yeast genome using a computational tool “Genome Carver”[2], and primers were designed automatically, conforming to the YeastFab Golden Gate assembly standard.

Manufacturing these 18,000 genetic parts across two continents poses several challenges: 1) Progress management and workload assignment: YeastFab is currently being undertaken by Edinburgh and Tsinghua Universities, where each team has multiple experimentalists simultaneously working on the project. Progress management is essential for the PIs to monitor the overall progress of the project, and thus can assign resources to workloads accordingly; 2) Statistical process control: manufacturing each of these 18,000 parts needs to go through 8 individual molecular biology operations (e.g., Polymerase Chain Reaction (PCR), ligation, transformation and sequencing), and at each step a rigorous quality control (QC) will be imposed to assess the success of the operation on biological samples. The success rate of passing each step needs to be systematically tracked to analyze the workflow; 3) Workflow management: to enable different experimentalists to work on different steps of the same batch of biological samples, we need to dissect the complex YeastFab workflow into trackable steps. The workflow management system will allow collaborative manufacturing, even across different sites. 4) Information capture and rendering: we need to capture information from multiple aspects, including the parts genetic information (e.g., sequence and original loci), the parts manufacturing history

as well as the parts functional characterization data. Rendering this information in a user-friendly manner will facilitate the design process in the future.

Herein we have developed YeastFabCAM, a computer assisted manufacturing platform to address these challenges in the YeastFab project. Computer Assisted Manufacturing (CAM) refers to the use of computational applications to assist in all operations of the manufacturing process. CAM is pervasive in mature manufacturing businesses, such as the car and electronics industries. CAM greatly improves the turnaround time, accuracy, consistency, and efficiency of the manufacturing process. To the best of our knowledge, we could not identify a CAM tool for biological parts manufacturing available as of today. The closest application is Taverna which is a workflow management system mainly for sequence analysis[4]. Our vision is to develop YeastFabCAM to formalize the DNA fabrication workflow and streamline the parts manufacturing process. This will be useful for similar large-scale parts manufacturing projects in the future.

2. IMPLEMENTATION AND AVAILABILITY

YeastFabCAM has been implemented using the Ruby on Rails web framework, and also uses PostgreSQL for the backend database. The part information is visualized and rendered by connecting to the ONION sequence editor (unpublished), which was developed using the D3.js JavaScript library. The sequencing verification step was described in [3]. YeastFabCAM is accessible at (<http://yeastfab.cailab.org>), and view-only accounts are available upon request.

3. YEASTFABCAM FUNCTIONALITIES

Progress tracking: YeastFabCAM currently provides two perspectives of viewing the project progress. The first one allows viewing the part manufacturing progress chromosome by chromosome, and the second one allows viewing by manufacturing sites (Figure 1A). This is useful for the PIs who track overall progress and make assignment decision.

Statistical process control: For each part which has passed

*Corresponding authors YC: yizhi.cai@ed.ac.uk and JD: jb-dai@biomed.tsinghua.edu.cn

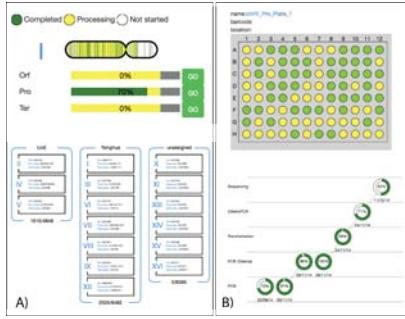


Figure 1: Progress tracking and statistical process control of YeastFabCAM. A) Upper: Progress view by chromosome; lower: Progress view by team. B) Upper: Sample progress in a 96-well plate format. Green: QC passed; yellow: in process; lower: Statistics of YeastFab workflow. Each step is assessed by a quality control, and the percentage refers to the success rate of passing the QC.

QC, the original cloning, testing and sequencing data will be traced in YeastFabCAM, and can be visualized by simply clicking the part name. In addition, all information is tracked by YeastFabCAM to perform statistical process control of the DNA manufacturing (Figure 1B). This data will be extremely valuable for future data mining exercises, such as to correlate the sequence motifs to the probabilities of experimental failure.

Part visualization and automated sequencing verification: we used our ONION sequence editor to visualize the part genetic information (Figure 2A) including sequence features, sequencing reads, and restriction sites. YeastFabCAM also allows users to batch upload Sanger ABI files and automatically verify the manufactured sequence fidelity with reference sequences (Figure 2B) [3]. The automated sequencing QC step liberates the experimentalists from visually inspecting sequencing trace files one by one, which is a tedious and error-prone process.

Part characterization information capture: YeastFabCAM captures not only the part manufacturing history, but also the functional characterization information of parts. We have characterized the transcriptional strength of 227 promoters using flow cytometry under various conditions such as oxidative stress (H_2O_2). Each characterization was performed in triplicate, and the information and statistics were recorded by YeastFabCAM.

4. FUTURE DIRECTIONS

Currently, YeastFabCAM relies on the PIs to assign the resources to the workloads, and in the near future we will be working on automatic assignment based on the availability of resources and the workload priority. We will also implement a 2D barcode sample tracking system in YeastFabCAM, so that it will serve as the Laboratory Information Management System (LIMS) for the future automation plan. Finally, we will develop an Application Program Interface (API) for the YeastFabCAM so that other Computer

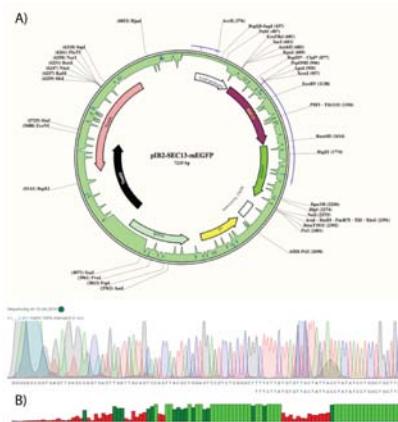


Figure 2: Parts visualization and automated sequencing verification. A) The sequence information is rendered using the ONION sequence editor. B) Automatic sequencing verification to identify mutations in the manufactured sequences.

Assisted Designers (CADs) will be able to access the functional characterization data of the YeastFab project.

5. ACKNOWLEDGEMENTS

YS, YS, ES and YC were supported by a Chancellor's Fellowship from the University of Edinburgh, and BBSRC grants BB/M005690/1 and BB/M025640/1 (to YC). JD is supported by Chinese Minister of Science and Technology grant 2012CB725201. We thank YeastFab team for beta testing YeastFabCAM and providing valuable feedback.

6. REFERENCES

- [1] Y. Guo, J. Dong, T. Zhou, J. Auxillos, T. Li, W. Zhang, L. Wang, Y. Shen, Y. Luo, Y. Zheng, J. Lin, G.-Q. Chen, Q. Wu, Y. Cai, and J. Dai. Yeastfab: the design and construction of standard biological parts for metabolic engineering in *saccharomyces cerevisiae*. *Nucleic Acids Res*, May 2015.
- [2] E. Scher, Y. Luo, A. Berliner, J. Quinn, C. Olgun, and Y. Cai. Genomecarver: harvesting genetic parts from genomes to support biological design automation. In *6th International Workshop on Bio-Design Automation*, 2014.
- [3] M. L. Wilson, Y. Cai, R. Hanlon, S. Taylor, B. Chevreux, J. C. Setubal, B. M. Tyler, and J. Peccoud. Sequence verification of synthetic dna by assembly of sequencing reads. *Nucleic Acids Res*, 41(1):e25, Jan 2013.
- [4] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Res*, 41(Web Server issue):W557–61, Jul 2013.

Successful Failure*: Best Practices for Quality Control of Large-scale DNA Assembly

Bryan A. Bartley

University of Washington
Department of Bioengineering
Box 355061
Seattle, WA 98195
bbartley@uw.edu

John H. Gennari

University of Washington
Biomedical & Health Informatics
Box SLU-BIME 358047
Seattle, WA 98195
gennari@uw.edu

Michal Galdzicki

Arzedo Corp
2715 W Fort St
Seattle, WA 98199
michal.galdzicki@arzedo.com

Herbert M. Sauro

University of Washington
Department of Bioengineering
Box 355061
Seattle, WA 98195
sauro@uw.edu

ABSTRACT

Quality control (QC) measures for large-scale DNA assembly are essential if synthetic biology is ever to succeed in engineering living systems of interesting complexity. While the DNA synthesis industry implements stringent quality control on synthesized oligos and genes[1], synthetic biology employs no consistent quality control practices for complex DNA assemblies. Moreover, synthetic biology employs a number of error-prone recombinant DNA methods, such as PCR and Gibson assembly, which may introduce or propagate errors at each step of a combinatorial assembly process[2]. Furthermore, once a synthetic construct is deployed in its production host, it may unexpectedly mutate and produce undesired behavior[3], [4]. Therefore, quality control must be practiced at each stage of assembly and should be periodically monitored during the actual deployment of a synthetic construct for a given application.

In these proceedings we propose guidelines for QC reporting for DNA assemblies, including a standardized visual schema that simplifies diagnosis of failures based on quality control data. These quality control data are encoded in the Synthetic Biology Open Language (SBOL)[5], a standardized data model and file format, so the information can be easily shared with and understood by downstream builders who must rely on properly verified upstream constructs. Finally, we demonstrate how our visual schema enables inspection of sequence quality across multiple scales of a complex, hierarchical design, making it possible to quickly pinpoint and diagnose failures.

In order to fulfill its promise, synthetic biology must overcome daunting limits to complexity, including unpredictable cellular environments, unexpected biomolecular interactions, and undesired mutations[6]. However, we can immediately overcome one important limit to complexity simply by establishing best practices for quality control of DNA assembly.

* Successful Failure was a term used to describe NASA's Apollo 13 mission, a complex engineering project plagued by unpredictable errors.

1. INTRODUCTION

Large-scale assembly of synthetic gene networks differs from gene synthesis. The upper limit in size for gene synthesis *de novo* is about 10^5 bases with a best-case error frequency of $1/10^5$ bases [7]. Thus constructs on the scale of 10^5 bp exceed a critical limit to complexity at which quality control becomes absolutely essential. In everyday practice at the synthetic biology bench, however, reality is much worse. Failure often occurs even for simple constructs. Often failures are due to errors in recombinant techniques like PCR and Gibson assembly[2], but also a variety of other factors such as human error, propagated construction errors, or mutations occurring after host transformation[4]. In order to isolate a successful construct, multiple clones must be screened, sequenced, and compared to the original design sequence. By one estimate 50% of gene synthesis costs are spent trying to isolate perfect clones[8]. DNA assembly failure is unavoidable, but mitigating its impact on the synthetic biology workflow would save money, and time.

2. QC DATA SHOULD BE REPORTED

After assembling a construct, its sequence must be verified, usually by Sanger sequencing. Sequencing data are compared to the target design, resulting in a sequence alignment or a multiple sequence alignment for large-scale constructs. Unfortunately, these data require expert analysis. Moreover, they are not kept in a centralized repository where other builders can access the data. Here we show how to turn sequence alignment data into an intuitive QC report in a standardized format that enables exchange and re-use of parts from repositories.

The Registry of Standard Biological Parts (partsregistry.org) greatly improved their quality control reporting by directly displaying sequence alignments for parts. We applaud these improvements. Here we advocate similar quality control best practices for both industry and academic synthetic biology. In addition we also demonstrate additional practices that enable QC on a large scale.

The specific sequence alignment metrics we recommend reporting are percent identity, percent coverage, and percent ambiguity. Identity quantifies the number of good and bad bases in a construct compared to its design sequence. Coverage is an

important metric for any construct of non-trivial length, because the construct may be incompletely sequenced. Cost is often a factor in deciding if a construct is completely sequenced. Ambiguity is important because noisy sequencing data may prevent us from confidently determining a construct's sequence. The Parts Registry reports metrics similar to identity, coverage, and ambiguity, but in terms of base length. As a matter of preference, we recommend reporting in percentages, because this quantifies our confidence in an immediately intuitive way.

3. QC SHOULD SUPPORT CORRELATED VIEWS OF CONSTRUCTION AND FUNCTION

Secondly, we recommend reporting QC metrics in a correlated view with a functional diagram of the construct (Figure 1). In essence, we propose a novel way of visualizing sequence alignment data. Inspecting an alignment base by base is comparable to examining machine code bit by bit, while our schema allows one to examine QC data from a comfortable level of abstraction. A correlated view of construction and function also allows the builder to quickly diagnose functional failures due to construction errors and mutations.

To illustrate this, we use SBOL Visual symbols to represent a biological design. In the following design, the functional units are a promoter, ribosome binding site, coding sequence, and transcriptional terminator. The QC statistics for a clone are derived from its sequence alignment and displayed below its design. This hypothetical construct has errors in the promoter consistent with mutational failures observed in practice[4]. If the construct exhibits loss-of-function, the promoter is likely to blame. An error in the RBS could also explain an observed failure, but this is inconclusive due to the ambiguity of its sequence.

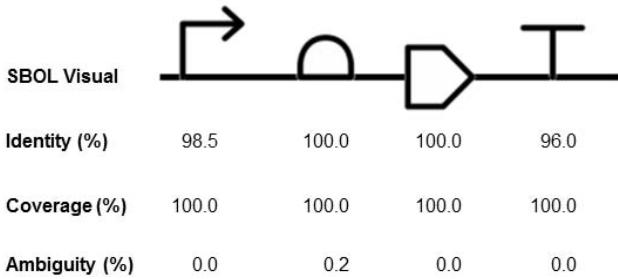


Figure 1. A correlated view of construction and function. The functional schematic was created with the SBOL Designer tool.

4. QC SHOULD BE SCALABLE

Our third recommendation for QC best practice is to support a scalable view of a construct. Multiple sequence alignments for large scale constructs are much too complicated to easily apprehend by eye. An additional advantage to representing sequence alignments schematically is that an arbitrarily large sequence can be collapsed into a single symbol. Thus, it is possible to view sequence alignment results at any scale of genetic organization, starting at the genome level and drilling down through subsystems, operons, genes, etc. As DNA assemblies scale in both structural and functional complexity, tools that support this QC best practice will become increasingly important.

5. INTERACTIVE TOOLS FOR QC

To highlight the quality control best practices described above, we will implement support for QC measures in the SBOL Designer [9] (Clark & Parsia LLC) plugin for the Geneious (Biomatters Ltd) sequence editor tool and the Tellurium [10] interactive Python platform for systems and synthetic biology. These tools leverage the open-source software libraries libSBOLj (Java) and pySBOL (Python) based on the standardized SBOL data model for representing biological designs. Additionally, we will demonstrate how multiple sequence alignments can be translated into SBOL using Sequence Ontology annotations. This allows us to directly associate QC data with a biological design, which is not possible with current alignment formats and yet another way the SBOL standard supports management of DNA repositories.

6. CONCLUSION

Synthetic biologists need better practices and better tools for managing QC. With current tools and practices, synthetic biologists have trouble managing QC even on a small scale. The best practices and interactive tools demonstrated here will help address this bottleneck in assembly and limit to complexity.

7. ACKNOWLEDGMENTS

Special thanks to Evren Sirin (Clark & Parsia LLC)

8. REFERENCES

- [1] W. Lint et al., “Easier DNA synthesis quality control with a semi-automated dimethoxytrityl cation assay.” *Biotechniques*, vol. 16, no. 3, p. 408, Mar. 1994.
- [2] D. G. Gibson et al., “Chemical synthesis of the mouse mitochondrial genome.” *Nat. Methods*, vol. 7, no. 11, pp. 901–903, 2010.
- [3] F. Ceroni et al., “Quantifying cellular capacity identifies gene expression designs with reduced burden.” *Nat. Methods*, no. February, pp. 1–8, 2015.
- [4] S. C. Sleight et al., “Designing and engineering evolutionary robust genetic circuits.” *J. Biol. Eng.*, vol. 4, no. 1, p. 12, 2010.
- [5] M. Galzicki et al., “The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology.” *Nat. Biotechnol.*, vol. 32, no. 6, pp. 545–50, 2014.
- [6] P. E. M. Purnick and R. Weiss, “The second wave of synthetic biology: from modules to systems.” *Nat. Rev. Mol. Cell Biol.*, vol. 10, no. 6, pp. 410–422, 2009.
- [7] S. Kosuri and G. M. Church, “Large-scale de novo DNA synthesis: technologies and applications.” *Nat. Methods*, vol. 11, no. 5, pp. 499–507, 2014.
- [8] S. M. Maurer et al., “Making Commercial Biology Safer: What the Gene Synthesis Industry Has Learned About Screening Customers and Orders.” *Goldman Sch. ...*, pp. 1–29, 2009.
- [9] SBOL Designer <http://clarkparsia.github.io/sbol/>
- [10] Tellurium <http://tellurium.analogmachine.org/>

Automated design of genetic logic circuits

Bryan S. Der
Massachusetts Institute of
Technology
Biological Engineering
bder@mit.edu

Douglas M. Densmore
Boston University
Electrical and Computer
Engineering
doug.densmore@bu.edu 3rd. author

Christopher A. Voigt
Massachusetts Institute of
Technology
Biological Engineering
cavoigt@gmail.com

Additional authors: Alec A.K. Nielsen, Jonghyeon Shin,
Prashant Vaidyanathan.

ABSTRACT

Living cells can sense and respond to changes in a variety of environmental signals. So far, engineering new information processing circuits to control these conditional responses has been a challenging and time-consuming process. We have developed a library of insulated genetic logic gates and a software design environment called Cello, which allow electronic design specifications to be automatically converted to a complete DNA sequence that executes the program in bacterial cells. Cello was used for automated design of 60 circuits, where 44 functioned correctly in the first experimental implementation. This result represents a significant advancement in the scale and success rate of genetic circuit design. To enable broad access, we implemented a web application (www.cellocad.org) where users can design logic functions of interest using an intuitive interface. Users also have the option to upload data using a constraints file describing custom sensors, logic gates, and actuators to build circuits in other experimental conditions and cell types of interest. We envision Cello providing a flexible and robust design environment for engineering circuits with diverse gates in diverse cell types.

1. SPECIFICATION

The Verilog hardware description language is used for high-level specification of circuit functions[4]. The synthesizable subset of Verilog can be mapped to a list of connected logic gates with physical implementations in hardware. Currently, Cello parses a subset of Verilog: case statements can specify a truth table, assign statements provide concise behavioral descriptions of combinational logic, and structural elements specify a gate-level circuit topology.

2. LOGIC SYNTHESIS

Verilog code is parsed to generate a truth table, which is the starting point for logic synthesis. The truth table is converted to a wiring diagram with gate types that are genetically available. This is done by first using a logic synthesis tool called ABC[1] to synthesize a minimized AND-Inverter Graph, consisting only of 2-input AND gates and NOT gates. Then, a NOR-Inverter Graph is generated using DeMorgan's rule: $(A \text{ and } B) = (\text{not } A) \text{ nor } (\text{not } B)$. Preferred logic motifs can then be substituted for functionally equivalent subcircuits. For example, we include an OUT-PUT OR motif and an optimal set of 3-input 1-output NOR

circuits in the motif library, which minimize circuit size upon substitution. The iterative subcircuit substitution routine can also incorporate other gate types such as AND, NAND, OR, and can constrain substitutions to adhere to the gate number and type constraints of the genetic gates library.

3. REPRESSOR ASSIGNMENT

Given the circuit diagram generated by logic synthesis, genetic gates from the library must be assigned to the gates in the circuit. We developed a set of NOT and NOR gates based on a set of prokaryotic repressors, which bind orthogonally[3], and are transcriptionally insulated at the 3' end by a ribozyme and at the 5' end by a terminator. Each gate has an experimentally measured and quantitatively unique response function fitted to a Hill equation, which relates an input value to an output value in the same standardized units (REU). Using a common signal (REU) allows levels to propagate through a multi-level circuit. Functional gate connections require the dynamic range of an upstream gate to span the threshold of a downstream gate[5] (Figure 1).

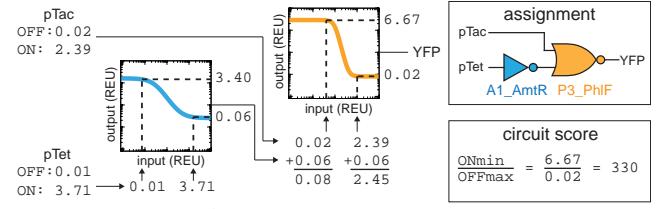


Figure 1: Response function matching. The pTac and pTet inputs have measured ON/OFF levels. The NOT gate (AmrR, blue) inverts the pTet according to the gate's Hill function. The pTac and pAmrR inputs for the NOR gate (PhiF, orange) are summed before applying the Hill function to calculate the output. The circuit score is computed as the ON/OFF dynamic range.

Repressor assignment identifies the optimal way to select and connect genetic gates to maximize the overall dynamic range for the circuit. The total number of possible assignments scales factorially as the circuit size and library size increase. Given a large and discrete search space, we use a Monte Carlo simulated annealing algorithm for assignment. The algorithm initializes by randomly assigning gates from the library to the gates in the circuit, and the Monte Carlo move swaps assignments at two gates. The first gate is randomly selected from the circuit, and the second gate is randomly selected from the circuit or the library. After a swap, the change in score and a temperature factor determine the probability of accepting the change. Thousands of swaps are performed in a single trajectory, and as the temperature

factor cools, the simulation converges. Multiple trajectories are run, and the circuit with the highest score is the output of the repressor assignment algorithm (Figure 2).

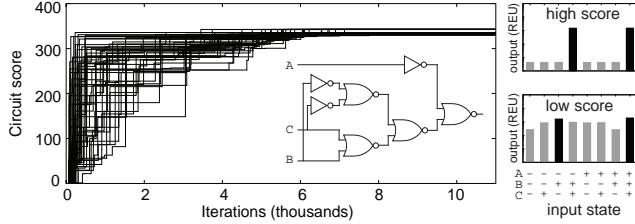


Figure 2: Simulated annealing for repressor assignment. 50 trajectories each with 10,000 move iterations (black lines) start at low scores but converge near the global optimum score (the dynamic range of the best gate in the library). The bar graphs show predicted outputs for a bad assignment and good assignment; ON and OFF states are shown in black and gray.

4. OUTPUT DISTRIBUTIONS

Given a repressor assignment, the performance of the circuit is predicted using cytometry data from experimental characterization of each response function. Response functions are characterized using discrete titrations of input levels, each producing an output distribution (Figure 3a). These discrete distributions are interpolated to generate a continuous probabilistic response function, which can compute an output distribution for any input value (Figure 3b). An input distribution can be converted to an output distribution by averaging all of the individual output distributions from each individual input value. By propagating distributions through each level in the circuit, the cytometry distributions of the circuit output are predicted (Figure 3c).

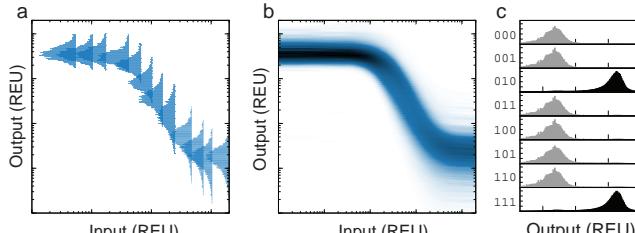


Figure 3: Probabilistic response function. (a) Discrete flow cytometry distributions for response function characterization. (b) Interpolation produces the continuous distribution-based response function. (c) Distributions are propagated through each circuit layer to predict the circuit output distributions.

5. PHYSICAL LAYOUTS

The final stage of the Cello design process is to generate physical part layouts that encode the assigned circuit. There are a combinatorial number of possible layouts for the same assignment: degrees of freedom include the order of tandem promoters in each gate, and the order and orientation of gates in the circuit. The Eugene language[2] is used to explore these degrees of freedom, and rules can be used to constrain the design space. For example, certain promoter orders are prohibited from being downstream of an adjacent promoter, and the gate orientations could be all forward, some reverse, or feature an alternating orientation pattern that may help reduce effects of terminator read-through. Rules can be added/removed to constrain/unconstrain the design space. In this work, a predetermined gate order and ‘all forward’ gate orientation was specified for an efficient Golden Gate cloning scheme.

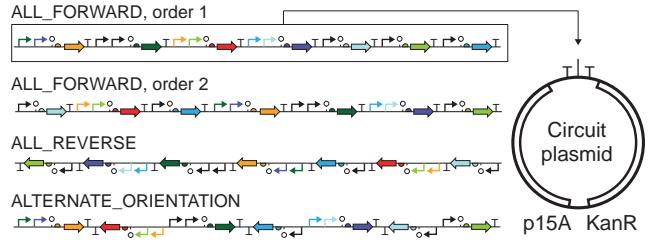


Figure 4: Combinatorial design of circuit layouts. The Eugene language uses rules for constrained combinatorial design of genetic constructs with varying part orders and orientations. A designed circuit module is inserted into the plasmid, and the complete DNA sequence is a Cello output.

6. EXPERIMENTAL RESULTS

Circuits ranging from 1 to 7 logic gates containing 27 to 55 genetic parts were designed with 44 of 60 circuits functioning correctly in all input states. These are unprecedented scales and success rates for circuit design. Prior to designing these circuits, significant experimental effort was required to avoid failure modes, including non-additivity of tandem promoters, terminator read-through, crosstalk between repressor:promoter pairs, and impaired cell growth from repressor expression. Rather than pursuing a convoluted computational model that attempts to precisely account for subtle contributions from each effect, we instead carefully curated the gates library and used empirically-determined design rules to minimize these effects. This strategy to maintain a simple computational model was validated by the high success rate, but the failure of 16 circuits is likely attributable to one or more of these effects.

7. USER CONSTRAINTS FILE

The ongoing goal for Cello circuit design is to standardize the software inputs such that users can specify their own preferred Boolean logic motifs, genetic gate library, circuit layout rules, and plasmid backbones or genomic locations. We use a highly specified constraint file to inform circuit design in Cello, and additional versions of the constraint file will allow Cello to design circuits in new experimental systems and cell types. Each system will be tied to the constraints file of that system, analogous to the choice of different microelectronic hardware to physically implement electronic designs.

8. REFERENCES

- [1] R. Brayton and A. Mishchenko. Abc: An academic industrial-strength verification tool. pages 24–40, 2010.
- [2] E. Oberortner, S. Bhatia, E. Lindgren, and D. Densmore. A rule-based design specification language for synthetic biology. *JETC*, 11(3):25, 2014.
- [3] B. C. Stanton, A. A. Nielsen, A. Tamsir, K. Clancy, T. Peterson, and C. A. Voigt. Genomic mining of prokaryotic repressors for orthogonal logic gates. *Nature chemical biology*, 10(2):99–105, 2014.
- [4] D. E. Thomas and P. R. Moorby. The verilog® hardware description language. 2, 2002.
- [5] F. Yaman, S. Bhatia, A. Adler, D. Densmore, and J. Beal. Automated selection of synthetic biology parts for genetic regulatory networks. *ACS synthetic biology*, 1(8):332–344, 2012.

Parameter inference for gene circuit models

Linh Huynh

Navneet Rai

Ilias Tagkopoulos

Department of Computer Science
& UC Davis Genome Center
University of California, Davis

{huynh,nnrai,itagkopoulos}@ucdavis.edu

1. INTRODUCTION

Parameter inference is crucial in any modeling effort. Parameter inference based on sequential fitting of data to each model leads to erroneous solutions due to over-fitting and ill-constrained parameter bounds. Parameter estimation through fitting multiple models simultaneously can reduce this error, albeit it is computationally intractable for most practical applications. Here, we propose an alternative approach of parameter inference cascades, where parameter values with low uncertainty are propagated to the sequentially fitted models. We propose how to deal with noise in the data and we introduce confidence intervals as a selection metric on parameter value propagation. We demonstrate how this approach reduces parameter estimation error in a synthetic circuit case-study.

2. METHODS AND RESULTS

Model: We use a simple model [1] that can capture both the processes of transcription and translation. More specifically, when a repressor R binds to a promoter p_R , the expression level of a gene g at the downstream of p_R is modeled by

$$\nu_g = \beta_{p_R} + \frac{\alpha_{p_R} - \beta_{p_R}}{1 + \left(\frac{\nu_R}{K_{p_R}}\right)^{n_{p_R}}} \quad (1)$$

where ν_g, ν_R are the expression level of g and R respectively, measured in relative expression units (REU) [2]. Parameters $\beta_{p_R}, \alpha_{p_R}, K_{p_R}, n_{p_R}$ represent the basal level, the promoter strength, the binding affinity, and its cooperativity respectively, pertaining to promoter p_R and its repressor R . In the case where a ligand L_R can bind and inactivate R , ν_R in equation 1 is updated by

$$\nu'_R = \frac{\nu_R}{1 + \left(\frac{[L_R]}{K_{L_R}}\right)^{n_{L_R}}} \quad (2)$$

where $[L_R]$ is the ligand concentration. Parameters K_{L_R} and n_{L_R} correspond to the dissociation constant and the Hill coefficient of the ligand, respectively.

A case study: Assume a cascade of repressors, as depicted in Figure 1. If we use the basic model, there are 16 parameters to capture all four circuits. For our evaluation, we fixed the parameter values, and then generated through simulations the corresponding synthetic datasets, on which we added a 10% of Gaussian noise. We also generated the data in triplicate and calculated the standard deviation of the output to simulate the experimental data in practice.

Model fitting: Suppose that each circuit C_i is modeled by

$$y_i = M_i(x_i, \theta_i) \quad i = 1, \dots, 4$$

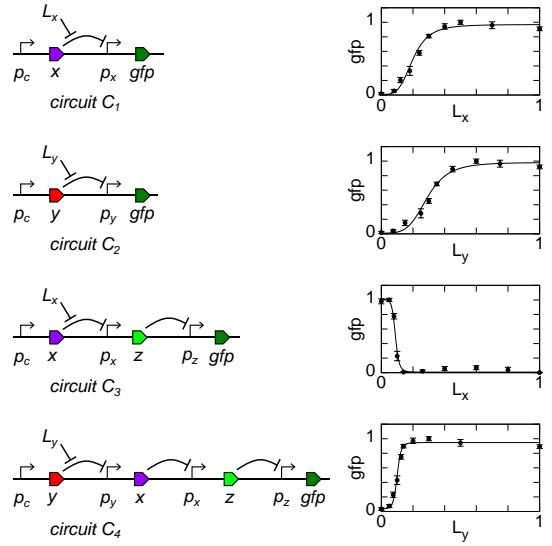


Figure 1: A case study with four cascade circuits (left) and their corresponding simulated data from a model with 10% Gaussian noise.

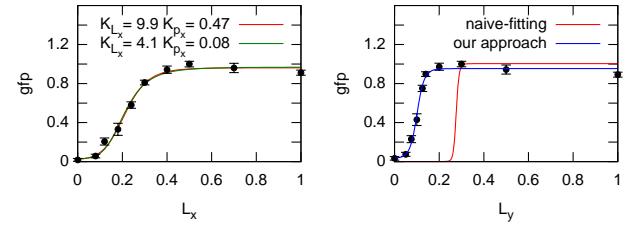


Figure 2: Problems with parameter inference. Multiple optimal solutions exist for the same circuit C_1 (left plot), optimal sequential parameter estimation from circuits 1-3 is unable to simulate points in circuit C_4 . The proposed method correctly identifies the optimal parameter set (right plot).

where x_i, y_i, θ_i represent the input, the output, and the set of model parameters, respectively, all for circuit C_i . For example, for the first circuit of Figure 1, x_i is the ligand L_x , y_i is the GFP concentration and θ_i is the set of six parameters that are needed in equations 1 and 2. Each parameter can appear in more than a single circuit, so we denote the set of all parameters $\theta = \bigcup_{i=1}^4 \theta_i$.

Let $D_i = \{(x_i^{(1)}, y_i^{(1)}, \sigma_i^{(1)}), \dots, (x_i^{(d_i)}, y_i^{(d_i)}, \sigma_i^{(d_i)})\}$ be a synthetic dataset with d_i data points of the circuit C_i and $\sigma_i^{(j)}$ capturing the standard deviation of the output $y_i^{(j)}$.

For each circuit C_i , if all data points are independent and the output value y_i has a Gaussian distribution then the log-likelihood [3] is

$$LL(D_i|\theta_i) = -\frac{1}{2} \sum_{j=1}^{d_i} \left(\frac{M_i(x_i^{(j)}, \theta_i) - y_i^{(j)}}{\sigma_i^{(j)}} \right)^2 + \text{const}$$

We can fit the value θ_i^* for parameters of each circuit C_i separately by solving the maximum likelihood problem

$$\theta_i^* = \underset{\theta_i}{\operatorname{argmax}} LL(D_i|\theta_i)$$

If the model M_i is sloppy [4], then two different parameter value combinations may have similar model outputs as in Figure 2. If we fit a sloppy model with a given dataset that contains noise, the fitted parameter values may be far from the actual values. To alleviate this, we can add more constraints on the parameters by fitting all the models simultaneously:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^4 LL(D_i|\theta_i)$$

However, solving this problem is computationally intractable due to the number of parameters involved. To reduce the computational cost, we can perform a sequential fitting, where the fitted parameter values are propagated to the next fitted model. However, any errors are also propagated and accumulated, which leads to erroneous solutions, as shown in Figure 2. To minimize this issue, we propose to propagate only parameter values of high confidence and introduce confidence intervals for this purpose.

Confidence interval We use the approach in [3] that is based on the profile likelihood [5] to estimate the confidence intervals of parameter values. The profile log-likelihood of a parameter $p_k \in \theta_i$ by fixing it to a value ν can be defined by

$$PLL_{p_k}(\nu) = \max_{\theta_i \in \{\theta_i | p_k = \nu\}} LL(D_i|\theta_i)$$

And the confidence interval for parameter p_k is

$$CI_\alpha(p_k) = \{\nu \mid -2PLL_{p_k}(\nu) \leq -2LL(D_i|\theta_i^*) + \Delta(\alpha)\}$$

where α is the confidence level. The threshold value $\Delta(\alpha) = icdf(\chi_1^2, \alpha)$ is the α -quantile of a χ^2 distribution with one degree of freedom.

Interestingly, by using this method the final prediction can be reliable even when its parameters have a large confidence interval, as it is the case in circuit C_3 . The combination of this ensemble learning and high-confidence parameter propagation is what leads to superior parameter inference results. Figure 3 depicts the solution to our case study by following Algorithm 1. Our approach can estimate the parameter value with a smaller error in all cases except for the parameter n_{L_x} , where both approaches are similar.

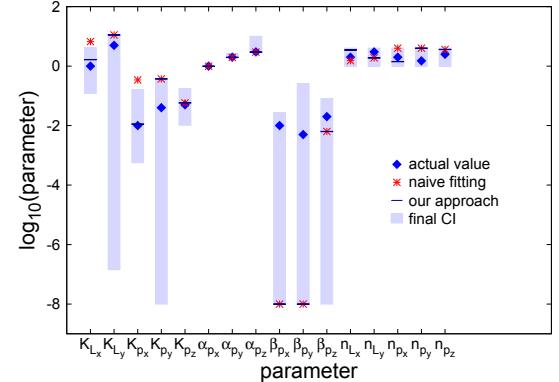


Figure 3: A comparison between actual and estimated parameter values.

3. DISCUSSION

We presented a new approach to infer the parameter value for multiple models with smaller error. Future work will be extension of this technique to more complex models and assessment of various optimization techniques such as symbolic computation or pattern search to reduce the computational cost.

Algorithm 1: Parameter inference

Input: Models M_i , datasets D_i , threshold values α, ε
Output: Parameter values and their confidence interval
begin
 $CI_\alpha(p_k) = Range(\nu_g) = (-\infty, +\infty) \forall p_k \in \theta, g \in C_i$
repeat
 for $i = 1 \rightarrow n$ **do**
 $G = \{g \in C_i \mid Range(\nu_g) < \varepsilon\}$
 $\theta' = \{p_k \in \theta_i \mid \exists g \in G \wedge p_k \text{ affects } \nu_g\}$
 $\theta'' = \{p_k \in \theta_i \mid CI_\alpha(p_k) < \varepsilon\}$
 $\hat{\theta}_i = \theta_i \setminus (\theta' \cup \theta'')$
 Fix value of ν_g ($g \in G$) and $p_k \in \theta''$ in M_i
 Estimate $\hat{\theta}_i$ by fitting M_i with D_i
 Update $CI_\alpha(p_k)$, $Range(\nu_g) \forall p_k \in \hat{\theta}_i, g \in C_j$
until $CI_\alpha(p_k)$ and $Range(\nu_g)$ do not change

4. REFERENCES

- [1] S. B. et al, “A synthetic multicellular system for programmed pattern formation,” *Nature*, 2005.
- [2] K. T. et al, “Refactoring the nitrogen fixation gene cluster from *klebsiella oxytoca*,” *PNAS*, 2012.
- [3] A. R. et al, “Structural and practical identifiability analysis of partially observed dynamical models by exploiting the profile likelihood,” *Bioinformatics*, 2009.
- [4] R. N. G. et al, “Universally sloppy parameter sensitivities in systems biology models,” *PLoS Comput. Biol.*, 2007.
- [5] D. V. et al, “A method for computing profile-likelihood-based confidence intervals,” *Applied Statistics*, 1988.

Design of Biological Circuits Using Signal-to-Noise Ratio

Jacob Beal

Raytheon BBN Technologies

10 Moulton Street

Cambridge, MA, USA 02138

jakebeal@bbn.com

1. MOTIVATION

Biological computing circuits have a role to play in many synthetic biology applications, such as precision cancer therapy, sensing chemical threats, or control of biosynthesis processes. Actually realizing such circuits effectively, however, has been quite difficult: until recently, neither high-precision prediction nor high-performance component libraries were available. Thus, although many design approaches for selecting components to realize a circuit have been proposed (e.g., [11, 6, 9], to name a few), it has been unclear which, if any, of these approaches was likely to actually be practical for the realization of biological circuits.

Recently, however, significant progress has been made in both circuit prediction and device performance. Calibrated flow cytometry [3] has enabled high-precision prediction of cascades and feed-forward circuits [5], as well as precision-design of resource competition systems [2]. At the same time, extensible families of high-performance devices have been created using four different architectures: TetR homologs [10], invertase logic [4], CRISPR-based repressors [7], and TALE-based repressors [8, 5].

Unfortunately, a signal-to-noise ratio (SNR) analysis of the actual properties of these device families shows that they do not yet correspond well with some of the digital logic assumptions that prior work on design approaches has relied upon. Instead, biological circuit design requires an approach that explicitly takes into account the degradation of a signal by each device in a computation, at least with the current families of available devices.

2. SIGNAL-TO-NOISE RATIO

From its inception, much of the work on biological circuits has embraced a digital logic paradigm. Key to realizing digital logic is for the amount of noise in the signal to improve from the inputs to the outputs of a device (generally via strong amplification). The amount of noise reduction—the “noise margin”—then determines the amount of noise that can be tolerated at each stage of computation without impacting the outcome of a computation of arbitrary complexity. Design tools for selecting devices to realize a circuit, such as MatchMaker [11] and SBROME [6], typically assume that there are devices available that provide noise reduction, and then attempt to select an appropriate set of such devices to realize the circuit.

We need to consider, however, whether such an assumption

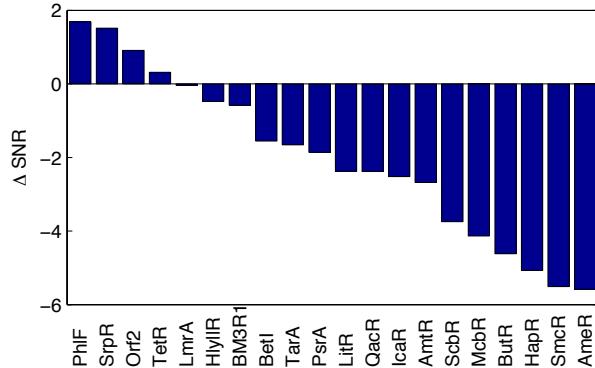


Figure 1: TetR homologs are the current best-performing logic device architecture: a few have positive maximum ΔSNR_{dB} , but most do not, and input/output levels do not generally match well between devices. Data reproduced from [1]

can actually be warranted. Mathematically, the relationship between signal and noise can be expressed as a signal-to-noise ratio (SNR), which may be computed using the standard formula: $SNR_{dB} = 20 \log_{10} \frac{A_{signal}}{A_{noise}}$ where A is the root-mean-square (RMS) amplitude of the signal and noise waveforms respectively. The efficacy of a logic device may then be expressed in terms of the difference between output and input SNR: $\Delta SNR_{dB} = SNR_{dB,out} - SNR_{dB,in}$. Any device with a significantly positive ΔSNR_{dB} can be used effectively to implement digital circuits; any other device degrades the signal that passes through it, limiting what computations are possible to implement. Moreover, the ΔSNR_{dB} that can actually be realized for a device depends on the levels and distributions of the inputs with which it is provided: a device that is positive when provided with well-matched inputs may be very negative when its inputs are instead too low or too high.

Characterization of synthetic biology devices and computations to date, however, has generally not actually analyzed signal to noise ratio, but instead provided only partial information, such as the ratio between “on” and “off” states, or the amplification of the device. While strong on/off ratio and strong amplification are generally necessary for good devices, they are not sufficient.

In fact, an SNR analysis of each of the current extensible high-performance device architectures, carried out in [1], reveals that none of them is currently known to be sufficient to implement complex digital logic circuits: TetR homologs [10]

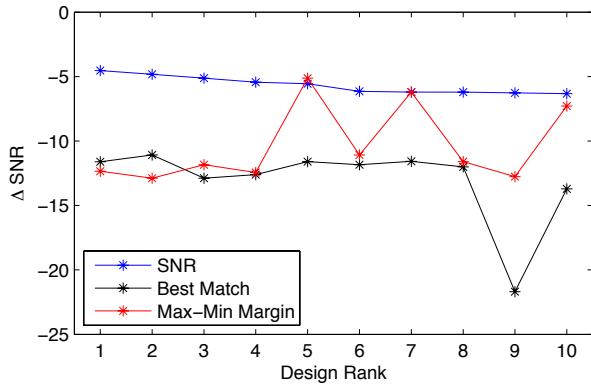


Figure 2: Design of a three-stage repressor cascade using TetR homolog devices shows that metrics based on digital assumptions are not effective at predicting signal degradation.

are currently the best-performing architecture, with a few devices providing the desired positive ΔSNR_{dB} for a narrow band of input values (Figure 1). They are highly heterogeneous, however, with most performing much more poorly, and generally poor matches between input and output levels. Invertase logic [4] has a sufficiently strong amplification, but its SNR performance is degraded by a significant non-responsive population. TALE-based repressors [8, 5] have insufficient amplification to support noise restoration. CRISPR-based repressors [7] may be better, but have only been characterized for on/off ratio, so amplification and input/output matching cannot yet be analyzed.

The effects of this insufficient ΔSNR_{dB} can be directly observed in the results reported for circuits constructed with these architectures. In every case [10, 5, 8, 7] the on/off ratio of the circuit output is much less than the on/off ratio of its inputs and earlier stages. This is symptomatic of a negative ΔSNR_{dB} , indicates that only simple and shallow circuits can currently be realized, and also indicates that digital logic noise restoration cannot be safely assumed.

3. SNR-BASED CIRCUIT DESIGN

Given the signal degradation of current biological computing devices, how do proposed approaches to design need to be adjusted? One option, of course, is to change nothing about design and just wait for devices with better ΔSNR_{dB} , but it is unclear how long this will take or to what degree it is even possible for large families of devices. More to the point, a great deal of interesting circuits can be implemented even with degrading signal strength, as is well demonstrated by the circuits in the same publications cited above.

To make principled decisions regarding the design of such circuits, we need a better metric that does not assume digital behavior. A reasonable choice for such a metric, of course, is simply SNR, since this directly measures the distinguishability of circuit outputs. For small circuits and libraries, this metric can be applied by brute force simulation of distributions. For example, Figure 2 shows the ΔSNR_{dB} ratings of the best ten designs for a three-stage inverter chain designed with TetR homologs from [10], beginning with an initial strong signal of low $10^{-1.5}$ and high $10^{1.5}$ a.u. and assuming a 2-fold standard deviation of per-cell expression.

Some of the choices are actually quite non-intuitive: for example, the top ten circuits include use of HlyllR, BM3R1, and PsrA, and the strongest repressor (PhlF) appears to be a poor choice for the first inverter, with the best circuit starting with PhlF being only -9.79 dB. Heuristics based on digital assumptions, however, such as maximizing the minimum noise margin [11] (using thresholds set at the 1:1 log slope), or maximizing input/output match quality, fail to accurately predict circuit performance and may select highly sub-optimal circuits.

Therefore, in order to realize effective biological circuit design using current signal-degrading devices, we can see that it is important to take the distribution of variation into account using a metric such as SNR. Heuristic and dynamic programming techniques that work for other metrics are likely to be adaptable for SNR as well, and there is also a considerable literature from the signal processing community that may be investigated for adaptability to the biological domain as well.

4. REFERENCES

- [1] J. Beal. Signal-to-noise ratio measures efficacy of biological computing devices and circuits. *submitted*.
- [2] J. Beal, T. E. Wagner, T. Kitada, O. Azizgolshani, J. M. Parker, D. Densmore, and R. Weiss. Model-driven engineering of gene expression from RNA replicons. *ACS synthetic biology*, 4:48–56, 2015.
- [3] J. Beal, R. Weiss, F. Yaman, N. Davidsohn, and A. Adler. A method for fast, high-precision characterization of synthetic biology devices. Technical Report MIT-CSAIL-TR-2012-008, MIT, April 2012.
- [4] J. Bonnet, P. Yin, M. E. Ortiz, P. Subsoontorn, and D. Endy. Amplifying genetic logic gates. *Science*, 340(6132):599–603, 2013.
- [5] N. Davidsohn, J. Beal, S. Kiani, A. Adler, F. Yaman, Y. Li, Z. Xie, and R. Weiss. Accurate predictions of genetic circuit behavior from part characterization and modular composition. *ACS Synthetic Biology*, 2014.
- [6] L. Huynh, A. Tsoukalas, M. Koppe, and I. Tagkopoulos. Sbrome: A scalable optimization and module matching framework for automated biosystems design. *ACS synthetic biology*, 2(5):263–273, 2013.
- [7] S. Kiani, J. Beal, M. R. Ebrahimkhani, J. Huh, R. N. Hall, Z. Xie, Y. Li, and R. Weiss. Crispr transcriptional repression devices and layered circuits in mammalian cells. *Nat. Meth.*, 11(7):723–726, 2014.
- [8] Y. Li, Y. Jiang, H. Chen, W. Liao, Z. Li, R. Weiss, and Z. Xie. Modular construction of mammalian gene circuits using tale transcriptional repressors. *Nature Chemical Biology*, 2015.
- [9] C. Madsen, C. Myers, T. Patterson, N. Roehner, J. Stevens, and C. Winstead. Design and test of genetic circuits using iBioSim. *IEEE Design and Test*, 29:32–39, 2012.
- [10] B. Stanton, A. Nielsen, A. Tamsir, K. Clancy, T. Peterson, and C. Voigt. Genomic mining of prokaryotic repressors for orthogonal logic gates. *Nature Chemical Biology*, 10(2):99–105, Feb. 2014.
- [11] F. Yaman, S. Bhatia, A. Adler, D. Densmore, and J. Beal. Automated selection of synthetic biology parts for genetic regulatory networks. *ACS Synthetic Biology*, 1(8):332–344, July 2012.

SynBad: A Synthetic Biology Design Framework

Owen Gilfellon

^{*}
ICOS

School of Computing Science
Newcastle University
o.giffellon@ncl.ac.uk

Paolo Zuliani

[†]
ICOS

School of Computing Science
Newcastle University
paolo.zuliani@ncl.ac.uk

Curtis Madsen

^{*}
ICOS

School of Computing Science
Newcastle University
curtis.madsen@ncl.ac.uk

Jennifer Hallinan

[†]
ICOS

School of Computing Science
Newcastle University
jennifer.hallinan
@mq.edu.au

Goksel Misirli

^{*}
ICOS

School of Computing Science
Newcastle University
goksel.misirli@ncl.ac.uk

Anil Wipat

[‡]
ICOS, School of Computing
Science, Centre for Synthetic
Biology and the Bioeconomy
Newcastle University
anil.wipat@ncl.ac.uk

ABSTRACT

In silico design is a fundamental component of the synthetic biology process. Tools for designing and exchanging genetic circuits are essential to support the design process and facilitate the transition from *in silico* design to *in vivo* implementation and testing. We have developed SynBad, an extensible design environment enabling the parts-based and model-driven design of genetic circuits. SynBad's modular design facilitates design reuse and the management of complexity. SynBad's architecture allows for the extension of functionality through the installation of plugins. Whilst supporting the manual CAD process, SynBad also supports genetic design automation using computational intelligence approaches. To this end, SynEA, an evolutionary algorithm-based tool for automating genetic circuit design was implemented as a plugin. SynBad is built on emerging open standards, including Standard Virtual Parts (SVPs) for representing designs, and the Synthetic Biology Open Language (SBOL) 2.0 for storage and exchange. In this work we describe the architecture of the SynBad system and demonstrate its functionality for the design of genetic circuits.

Keywords

Genetic circuits, SVPs, CAD, SBOL

1. INTRODUCTION

A major aim of synthetic biology is to enable the engineering of complex and novel biological systems. Computer-aided design (CAD) is a common approach in other engineering disciplines for managing complexity, where routine design tasks can be automated and abstracted. In Integrated Development Environments for computer programming, for example, the burden on programmers is lessened by providing re-usable, tested, higher-level abstractions more suited to human-manipulation, while automating the mapping from abstract to lower-level implementation of designs. Synthetic biology design, however, has not yet reached the stage where the forward engineering of biological systems is routine [4].

*Interdisciplinary Computing and Complex BioSystems

[†]Currently at Macquarie University

[‡]To whom correspondence should be addressed

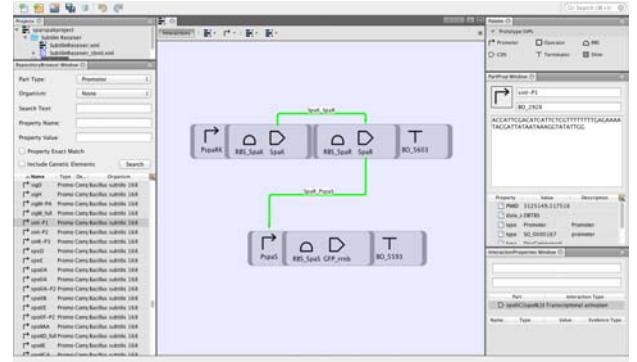


Figure 1: An editor view in SynBad.

2. SYNBAD

Complex synthetic biology designs require computational approaches to their construction and optimisation. A number of software design tools have been developed to meet this challenge [7, 1]. SynBad also utilises the concept of a CAD system that is common to many previous approaches but builds on the approach offered by others with the addition of three major features; (i) SynBad enables a modular model-based design approach to genetic circuit design, promoting module re-use, with the ability to simulate designed circuits (ii) the inclusion of computational intelligence-based approaches for circuit design and optimisation (iii) a modular plugin-based framework to support third-party software extensions. SynBad also supports emerging standards such as SBOL 2.0.

2.1 SVP-based Design

The parts for the SynBad system are based on Standard Virtual Parts. SVPs are composable models of physical biological parts and their interactions, represented using XML. SVPs are abstract models, which are parameterised from templates representing genetic parts such as promoters and protein coding sequences, and their interactions [6]. SVPs may also contain information about the sequence of the parts they represent. SynBad designs can use any combination

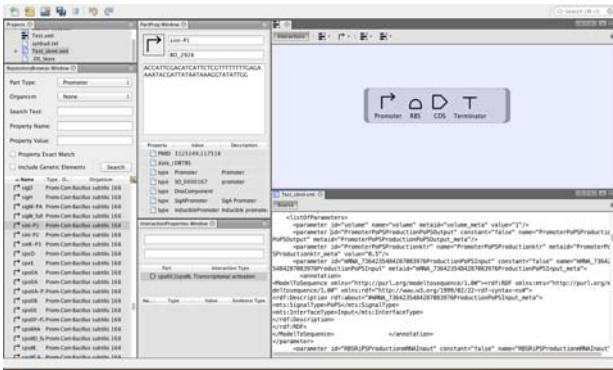


Figure 2: A prototype transcription unit compiled to SBML.

of SVPs specified either abstractly with hypothetical or desired properties, or concretely, with preinitialised parts from a Web-service based parts library. Such part libraries can be populated using parameters drawn from the literature, or from *in vivo* or *in silico* experiments. SynBad is currently configured to use the Flowers Virtual Parts Repository¹ (VPR), an SVP repository maintained by the Interdisciplinary Computing and Complex BioSystems (ICOS)² research group at Newcastle University. The VPR contains 2226 parts and interactions, initially produced from an integrated dataset [5]. SynBad supports modular design. SVPs are hierarchically organised into SBOL2.0 compatible modules, composed from parts and other modules. This approach enables any design element in SynBad, from a single part to entire design, to be reused as a module in another design.

2.2 Visual Interface

SynBad has a visual design interface. Editor views present abstractions of the designs, emphasising or de-emphasising different aspects of each design, while operating on the same underlying data model. Views abstract the process of manipulating the SVPs composing the design, and allow synthetic biologists to share a common design language based on the manipulation of standard parts and interactions. SynBad then automates the compilation of these part-based designs into simulatable Ordinary Differential Equation models in the Systems Biology Markup Language (SBML) format by composing the underlying SVPs into a single model.

2.3 Extensibility

SynBad is open source, with an extensible architecture allowing additional functionality to be installed using plugins. SynBad designs are stored using the Synthetic Biology Open Language (SBOL) [2], with additional annotations for platform-specific data, such as SVPs. SynBad offers synthetic biology tool developers a common environment for SBOL and SVP-based tools. Emerging open source standards such as SBOL and SVPs were chosen in order to maximise interoperability with other tools. Plugin developers can take advantage of the APIs provided, including those for storing, manipulating, and composing SVPs, and expose

¹<http://www.virtualparts.org/>

²<http://ico2s.org>

their own APIs. Plugins can use SVPs to represent their designs, or developers can design their own mappings from the SBOL documents while benefitting from the common design environment SynBad offers.

2.3.1 SynEA - A SynBad Plugin

The extensibility of the framework is demonstrated by the SynEA plugin, an automated design tool for SynBad based on an evolutionary approach [3]. SynEA can generate networks *de novo*, or use designs implemented in SynBad as initial designs. SynEA can also optimise existing designs, and fit abstract networks with parts from repositories. Designs produced by SynEA are added to the user's project, for export or visualisation and manipulation in the editor views.

3. FUTURE WORK

SynBad and SVPs are ongoing, open-source projects, and we encourage feedback from the community to help guide development. In the short term, we plan to expand SVPs, and SynBad, with support for more modelling formalisms, such as rule-based modelling, the development of workflows enabling more sophisticated feedback between *in silico* and *in vivo* synthetic biology approaches. Additionally, SynBad will include plugins for design verification, a promising feature of model-based design.

4. AVAILABILITY

The Java source code for SynBad is available under the Apache License at:

<http://ico2s.org/software/synbad.html>.

References

- [1] D. Chandran et al. TinkerCell: modular CAD tool for synthetic biology. *Journal of Biological Engineering*, 3:19, Jan. 2009.
- [2] M. Galzicki et al. The synthetic biology open language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nature biotechnology*, 32(6):545–550, 2014.
- [3] J. Hallinan et al. Tuning receiver characteristics in bacterial quorum communication: An evolutionary approach using standard virtual biological parts. In *Computational Intelligence in Bioinformatics and Computational Biology, 2014 IEEE Conference on*, pages 1–8. IEEE, 2014.
- [4] J. T. MacDonald et al. Computational design approaches and tools for synthetic biology. *Integrative biology: quantitative biosciences from nano to macro*, 3(2):97–108, Feb. 2011. PMID: 21258712.
- [5] G. Misirli et al. Bacillondex: An integrated data resource for systems and synthetic biology. *Journal of Integrative Bioinformatics*, 10(2):224, 2013.
- [6] G. Misirli et al. Composable modular models for synthetic biology. *ACM Journal on Emerging Technologies in Computing Systems*, 11(3):22:1–22:19, Dec. 2014.
- [7] C. J. Myers et al. iBioSim: a tool for the analysis and design of genetic circuits. *Bioinformatics*, 25(21):2848–2849, Nov. 2009.

D-VASim: Dynamic Virtual Analyzer and Simulator for Genetic Circuits

Hasan Baig and Jan Madsen

Department of Applied Mathematics and Computer Science
Technical University of Denmark
[{haba, jama}@dtu.dk](mailto:{haba,jama}@dtu.dk)

1. INTRODUCTION

A genetic circuit represents a gene regulator network that is triggered by a combination of external signals, such as chemicals, proteins, light or temperature, to emit signals to control gene expression or metabolic pathways accordingly. In order to match the intended behaviour, genetic circuits are either assembled from a standard library of well-defined genetic gates or from parts of an available library, for instance, BioBricks. The obtained behavior can be validated through in-silico analysis, solving reaction kinetics using ordinary differential equations (ODEs) or by stochastic simulation, with the aim to reduce the number of required in-vitro experiments.

We present a behavioural simulation and analysis tool that allows the biologist to carry out virtual lab experiments as an *interactive* process during simulation of the genetic circuit, rather than a batch process, which is current practice. We believe that this increases the insights gained from the analysis and allows for exploring more parameters in an intuitive manner.

2. GENETIC CIRCUIT ANALYSIS

The Systems Biology Mark-up Language (SBML) is a standard way of representing computational biological models [1]. It is a machine-readable format, which enable models to be shared and published in a form that can be used by different software tools.

Beside the functional behavioural of the biological systems, SBML allows the user to model a sequence of input patterns in order to capture a more elaborate experiment. This is done through *events*, which describe the instantaneous, discontinuous state changes in the model [1]. For example, in genetic circuits, *events* are used to trigger the concentration of any input species to a certain level, at a specific point in time, and to observe the effects on the concentration of output species. Since *events* are predefined, they cannot be changed during runtime, which means that the output of a genetic circuit can be observed only for defined events. In order to observe the output, the different set of input conditions, i.e., when to change what input to which level, must be defined in each event. Even for moderate sized genetic circuits, capturing all combinations of inputs and concentration levels may require a very large number of events to be defined and simulated.

The ability to interact with the model, during runtime, makes it more convenient to observe the behaviour and directly make changes of input species as a reaction to the observed changes. This not only helps the user to analyse the model appropriately by triggering the concentration of input species to any level and at any instant of time, but it also makes the user free of defining long list of events for all the possible combinations of inputs in the SBML description.

There are more than 260 systems biology tools [2], which assist users in model construction and analysis. Some of these tools

serve as a toolbox for commercial platforms including MATLAB, Mathematica, and Oracle; some are developed as APIs or plugins to specific software systems, while the rest are independent tools for design and simulation. A vast majority of these tools supports reading and/or writing SBML files. To the best of our knowledge, there exist no tools that allow users to trigger/change input species on the fly during the simulation, effectively creating a *virtual lab*.

3. VIRTUAL LAB SIMULATION

In the wetlab, the biologists are either provided with ready-made biological models available in test tubes or are given a specification/recipe from which to prepare it in the lab. Their duty is to analyse the model and verify its functional behaviour. This analysis is done interactively by among other things, increasing the molar concentration of input species at any instant of time and observing the effects.

This motivated us to develop a virtual laboratory environment where users can perform interactive experiments by varying the molar concentrations during run time. This inspiration lead us to develop D-VASim (Dynamic Virtual Analyzer and Simulator), a user-friendly environment to simulate and analyse the behaviour of genetic circuit models written in SBML. D-VASim takes as input a SBML file and generates an interactive virtual instrument (VI) to simulate the behaviour of the biological model. This virtual instrument works as a standalone simulation tool for the particular SBML model. Currently D-VASim offers two types of virtual instruments, one based on solving reaction kinetics using ODEs, and the other based on stochastic simulation.

Both D-VASim and the generated virtual instrument are developed on National Instruments LabVIEWTM¹ platform, which is a graphical programming platform commonly used to rapidly develop instrumentation systems for data acquisition, instrument control, and industrial automation [3].

Besides giving the biologist the feeling of being in the lab, D-VASim has also proved useful to help early-stage researchers or students, with little experience in biology, to get an intuitive feeling of the underlying biological processes and their interactions. A virtual laboratory environment is desired for such inexperienced users to observe the live biological phenomenon by varying the species concentrations without being afraid of crossing the threshold values.

D-VASim also allows user to analyse the SBML model components. Depending on the parameter settings, D-VASim generates a VI for deterministic or stochastic analysis separately. Once the instrument is generated, the user can analyse the model by varying those species' concentrations, which acts as external modifiers. This makes the VI more analogous to the real-life experimentation where the operator can increase the molar

¹ LABoratory Virtual Instrument Engineering Workbench

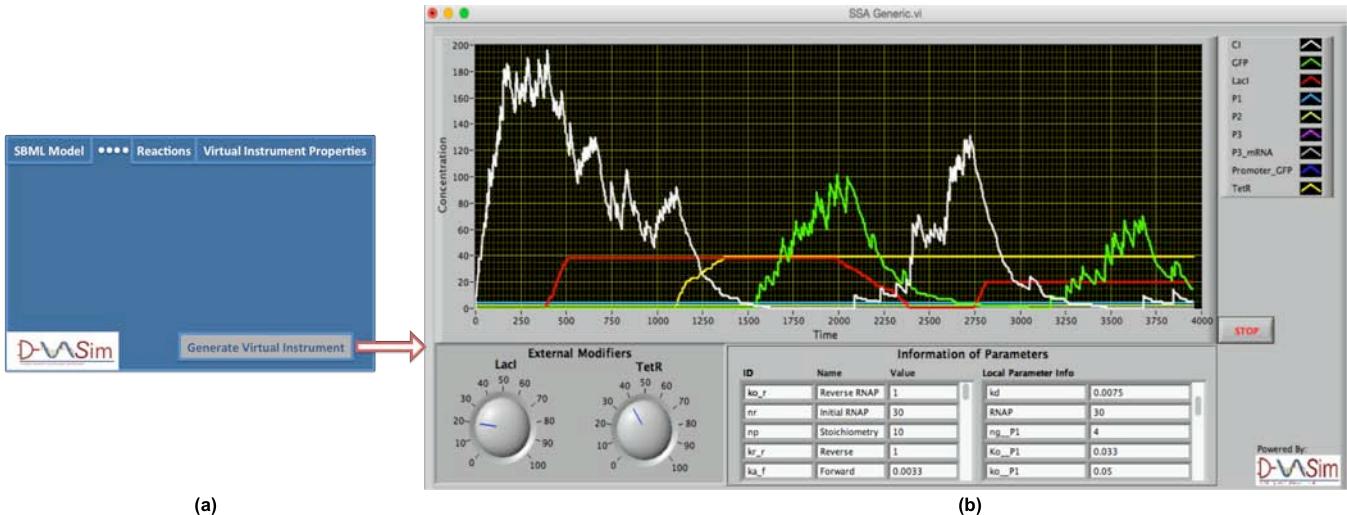


Figure 1. D-VASim (a) Top-level diagram of D-VASim showing different tabs (b) Generated virtual instrument for stochastic simulation of genetic AND gate model.

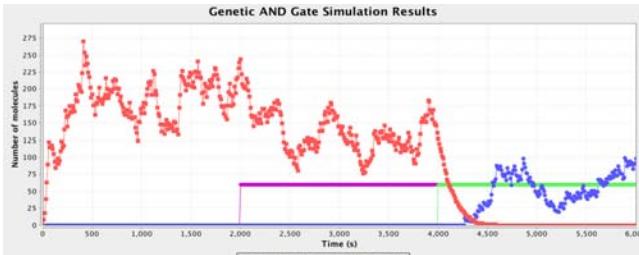


Figure 2. Static stochastic simulation plot of genetic AND gate generated by iBioSim [5].

concentration of external inputs only. Unlike the real-life experimentation, where the reaction takes place at specific rates, users can speed-up or slow-down the reaction by varying the parameter values during runtime.

4. EXPERIMENTAL RESULTS

D-VASim is tested with different example models imported from existing tools including CellDesigner [4] and iBioSim [5]. We have also tested some of the genetic gates modelled by Myers [6]. Due to the space limitations, only the results of a genetic AND gate [6] is included here.

Figure 1 shows a screen captures of D-VASim running the genetic AND gate model. Figure 1(a) shows the basic top-level diagram of D-VASim containing different tabs. For example, *Reactions tab* helps the user to analyse the reaction kinetics of the model in a user-friendly manner. Similarly, *Virtual Instrument Properties* tab allow users to set up different properties of the VI including the VI window-bounds, sizes of VI objects (knobs, graph-window etc.), deterministic or stochastic simulation, timing bounds for ODE simulation, type of continuous solver for ODE simulation etc. After setting up these properties, the VI can be generated by pressing the button *Generate Virtual Instrument* depicted in Figure 1(a). Figure 1(b) shows the virtual instrument generated for stochastic simulation of the genetic AND gate model [6]. The screen of the VI is captured during the simulation, which clearly shows the interactive stochastic simulation results. In comparison, Figure 2 illustrates the static stochastic simulation results of the same genetic AND gate generated by iBioSim [5]. As shown in Figure 2, the events, to trigger the number of molecules of TetR and LacI to 60, are predefined to be activated at time 2000 and 4000 units respectively. Also, the simulation runs for a predefined interval, 6000 time units in this example. From Figure 2, it can be

observed that the production of GFP (blue curve) starts when the number of molecules of both the inputs, TetR and LacI, reaches at the same level i.e. 60. It is, however, more evident in Figure 1(b) that the production of GFP (green curve) starts even when the concentration of both the inputs are not same (at time unit 3200). Therefore, to observe the behaviour of combinatorial genetic logic circuits more clearly, either all the possible combinations of inputs with all possible concentration levels should be defined in the list of SBML events – a tedious task, or the model's behaviour should be examined in the interactive environment. It may also be possible to generate the list of SBML events by running pre-written scripts with minimal efforts, but the idea of interacting with the model during runtime gives the insight of performing live virtual lab experiments. Hence the significance of a run-time interactive simulation environment, like D-VASim, is more obvious as it helps the user to analyse the model more easily and explore its parameter space intuitively.

5. SUMMARY

We are currently working on an algorithm to make the D-VASim capable of extracting the Boolean expression from the interactive simulation. It will help students and scientists to validate if the genetic circuit model behaves as expected. In future, we plan to incorporate a Boolean logic minimization tool for genetic cost reduction, which will specifically be helpful when building cascaded genetic circuits.

6. REFERENCES

- [1] The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core, October 06, 2010.
- [2] Systems Biology Mark-up Language Software Matrix, http://sbml.org/SBML_Software_Guide/SBML_Software_Matrix.
- [3] NI LabVIEW, <http://www.ni.com/labview/>.
- [4] Funahashi, A.; Matsuoka, Y.; Jouraku, A.; Morohashi, M.; Kikuchi, N.; Kitano, H. "CellDesigner 3.5: A Versatile Modeling Tool for Biochemical Networks" Proceedings of the IEEE Volume 96, Issue 8, pp. 1254 – 1265 Aug. 2008.
- [5] C. Madsen, C. Myers, T. Patterson, N. Roehner, J. Stevens, and C. Winstead, "Design and test of genetic circuits using iBioSim," IEEE Design and Test, 29(3): pp. 32-39, May/June 2012.
- [6] Chris J. Myers, "Engineering Genetic Circuits", Chapman & Hall/CRC Press, July 2009.

Fluigi: An Automated Framework for Creating Bioelectronic Devices

Haiyao Huang, Aaron Heuckroth, Ryan J. Silva, and Douglas Densmore

Boston University
Boston, MA 02215
dougd@bu.edu

ABSTRACT

One goal of synthetic biology is to design and build genetic circuits in living cells for a range of applications. Major challenges include increasing the scalability and robustness of engineered biological systems and streamlining and automating the synthetic biology workflow of specify-design-assemble-verify. We present a novel hardware/ software/ wetware framework, Fluigi, that allows for functional specification to be “synthesized” into genetic networks that are spatially arranged and connected by a scheduled interconnection network. This arrangement is created via 3D-printable microfluidics and augmented with a library of electronic actuators, sensors, and controllers. This framework will push the boundaries of hybrid bio-electronic integration and will enhance the current state of synthetic biology design automation by introducing an open, accessible, and democratized method of laboratory customization.

1. INTRODUCTION

The goal of synthetic biology is to use naturally existing constructs in biology (such as repressible and inducible genes) for novel applications [Cameron et al. 2014]. Microfluidics can assist researchers by reducing reagent use, increasing throughput and automation, and precisely controlling the spatial and temporal environment of their experiments [Huang and Densmore 2014]. However, adoption of microfluidics in synthetic biology labs has been slow due to the expertise and equipment needed to design and manufacture microfluidic devices [Ferry et al. 2011]. We present here Fluigi, a framework and toolchain for automating the design and manufacture of microfluidic devices for synthetic biology through the use of 3D printing and CAD.

The Fluigi framework, shown in Figure 1, starts with the specification of a function in a description language (Verilog) and a set of physical design rules. Fluigi consists of three main blocks: the hardware for microfluidic valve control, remote communications, and sensors (Channel Chomp), CAD for microfluidic devices (Piranha Planner), and a 3D printing setup for printing molds for microfluidic devices (3DuF). A different tool converts the design specification into a set of biological devices that implement the function [Nielsen et al.]. Fluigi takes the set of biological components and the constraints to layout a microfluidic device that houses and monitors those components. It also produces photomasks for use with multilayer soft lithography or a 3D model for fabrication with consumer 3D printers and the valve control sequences to operate the microfluidic device.

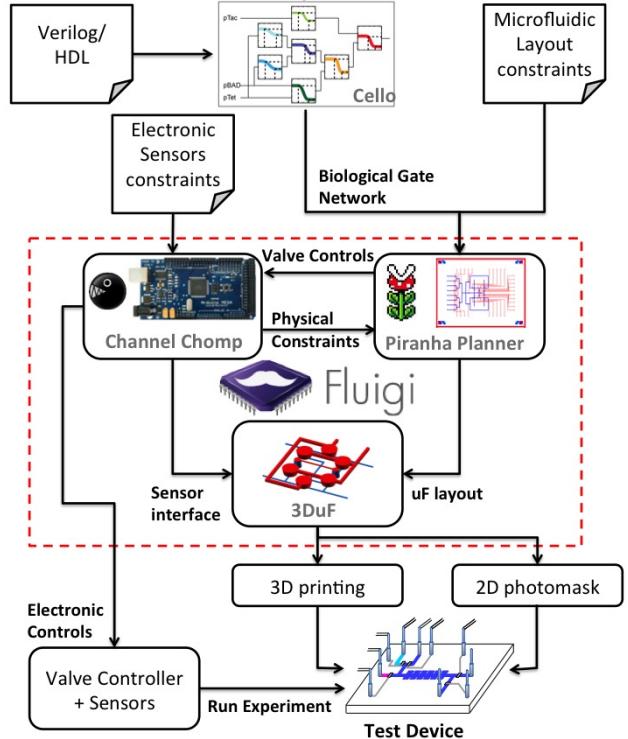


Figure 1: The three main blocks of Fluigi: Channel Chomp for hardware control and chip interfaces, Piranha Planner for microfluidic CAD, and 3DuF for rendering and fabricating both 2D and 3D molds for microfluidic devices.

2. CHANNEL CHOMP

Accurate, real-time monitoring and dynamic control are challenges in the creation of microfluidic devices. Channel Chomp is a modular approach to delivering these capabilities to the device under test. Potential applications are shown in Figure 2. Designers can define input and output constraints for assay instrumentation using open-source hardware and software solutions. Valve control schemes are compiled into an executable experimental protocol automated through the use of parametrized, 3D-printed, pneumatic control devices. The use of open-source software and inexpensive commercial off-the-shelf hardware creates an environment for electronic analysis and control that is fully-accessible, affordable and easily reproduced.

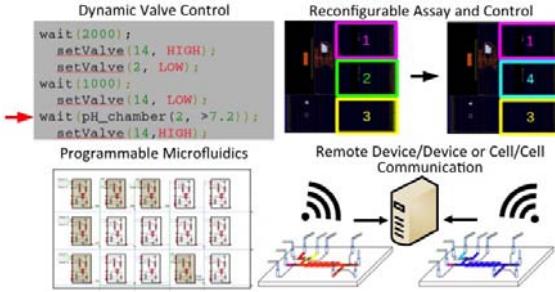


Figure 2: Applications for Channel Chomp include dynamic valve control, reconfigurable assays, programmable microfluidics, and remote communications between experiments.

3. PIRANHA PLANNER

Piranha Planner is the CAD software responsible for the physical description and layout of the microfluidic device being designed. The workflow for this process is shown in Figure 3. A device is represented as a data structure where each layer is a graph of channels and components such as ports, valves, and cell traps. A layout is generated from a set of physical constraints and a netlist describing the connections in the device using simulated annealing for placement [Betz and Rose 1997] and Hadlock’s algorithm [Hadlock 1977] for routing. If given a set of biological devices that perform a function, Piranha Planner will generate a design of a device to house the biological devices and a set of valve control patterns to allow intercellular communications between them.

4. 3DuF

The adoption of consumer 3D printing in laboratory environments has led to increasingly sophisticated devices that can be fabricated using commodity hardware [Takahashi et al. 2014]. We present here a process for printing molds for microfluidic devices using an off-the-shelf 3D printer. Designs are generated by using Python scripts to place primitive features and configure device layers. These scripts generate header files for an OpenSCAD library which converts a list of primitives and parameters into a printable 3D model for each layer of the device. Each model is converted into instructions for a 3D printer using standard slicing software, taking into account nozzle size, minimum layer height, and

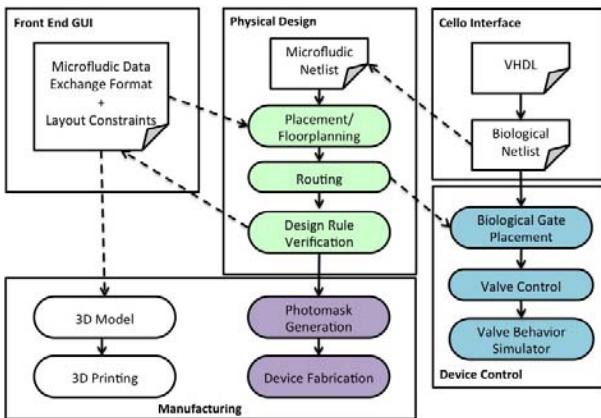


Figure 3: The workflow for Piranha Planner, from physical design to manufacturing and device controls.

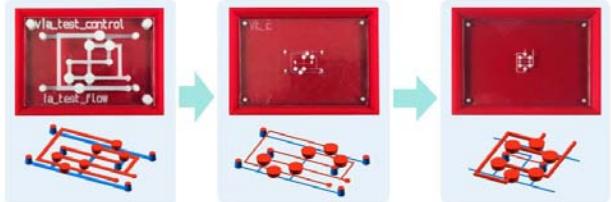


Figure 4: An initial prototype of a parametric microfluidic transposer module was designed and fabricated using 3DuF. The design was refined over the course of a week to dramatically reduce both individual feature size and the overall footprint.

other printer-specific factors. Features are then printed directly onto glass slides and assembled for PDMS casting. We show in Figure 4 a prototype of a parametric transposer module designed with this process that can be easily customized and adapted for fabrication at a wide range of feature sizes.

5. CONCLUSION

The integration of synthetic biology, 3D printable microfluidics, and electronics has the capability to increase the scale of engineered biological systems for applications in cell-based therapeutics and biosensors, expand on the idea of distributed biological computation, and produce new rapid prototyping platforms for the characterization of genetic devices. The combination of 3D printing for manufacturing coupled with off-the-shelf electronics can increase the use of microfluidics in synthetic biology and promote opportunities for interdisciplinary research and collaboration.

6. REFERENCES

- [Betz and Rose 1997] Vaughn Betz and Jonathan Rose. 1997. VPR: A new packing, placement, and routing tool for FPGA research. In *International Workshop on Field Programmable Logic and Application*.
- [Cameron et al. 2014] D Ewen Cameron, Caleb J Bashor, and James J Collins. 2014. A brief history of synthetic biology. *Nature Reviews Microbiology* 12, 5 (2014), 381–390.
- [Ferry et al. 2011] MS Ferry, IA Razinkov, and J Hasty. 2011. Microfluidics for synthetic biology from design to execution. *Methods Enzymol* 497 (2011), 295.
- [Hadlock 1977] FO Hadlock. 1977. A shortest path algorithm for grid graphs. *Networks* 7, 4 (1977), 323–334.
- [Huang and Densmore 2014] Haiyao Huang and Douglas Densmore. 2014. Integration of microfluidics into the synthetic biology design flow. *Lab on a Chip* (2014).
- [Nielsen et al.] Alec A.K. Nielsen, Bryan S. Der, Jonghyeon Shin, Prashant Vaidyanathan, Douglas Densmore, and Christopher A. Voigt. Genetic circuit design automation. *In Submission* (????).
- [Takahashi et al. 2014] Chris N Takahashi, Aaron W Miller, Felix Ekness, Maitreya J Dunham, and Eric Klavins. 2014. A low cost, customizable turbidostat for use in synthetic circuit characterization. *ACS synthetic biology* 4, 1 (2014), 32–38.

Apprentyeast: a command-line synthetic yeast architect's apprentice.

Laura Adam, PhD
Department of Electrical
Engineering
University of Washington
Seattle WA
ladam@uw.edu

Nick Bolten
Department of Electrical
Engineering
University of Washington
Seattle WA
nbolten@gmail.com

Eric Klavins, PhD
Department of Electrical
Engineering
University of Washington
Seattle WA
klavins@uw.edu

ABSTRACT

Apprentyeast aims to provide an abstraction layer between the circuits designed by synthetic biologists and their implementation as a synthetic yeast strain constructed in the lab. Apprentyeast is a command line tool implemented in python that relies on Aquarium, which is a human-in-the-loop lab automation system that includes a Laboratory Information Management System (LIMS), running in the Klavins Lab, and gets data from public sources such as the Saccharomyces Genome Database.

Apprentyeast can be considered as an architect's apprentice. It is handling the wet lab realization of high-level synthetic designs on the behalf of the organism designers while learning the best practices from experts. For instance, Apprentyeast can be used to knockout a gene. The organism designer simply inputs the name or id of the yeast gene they desire to knock out and the name of the selection marker to insert. Apprentyeast will then retrieve the sequences to design the corresponding primers and fragments, load them into Aquarium through a web API for automated construction. Being an apprentice, Apprentyeast learns the most commonly used construction strategies in the lab and preferentially uses them. We looked at all the Gibson assemblies performed through Aquarium in order to infer rules that then serve to design new plasmids.

Categories and Subject Descriptors

D.2.2 [SOFTWARE ENGINEERING]: Design Tools and Techniques

General Terms

Algorithms, Management, Design, Experimentation, Human Factors, Standardization, Languages, Theory, Verification.

Keywords

supervised learning, yeast, synthetic biology, plasmid, data mining.

1. Background

Synthetic biology has been tackling the problem of reproducibility in biology by standardizing wet lab practices and recording experiments [1].

In Klavins Lab, researchers have been using Aquarium [2] to formalize and automate protocols. Wet lab work is performed and logged by technicians using a touch-screen interface. Aquarium has been running for over a year providing us access to wet lab usage.

For instance, it is not clear what the optimal recipe to build a plasmid is using Gibson Assembly given an extended and shared library of fragments. Our lab members use a modular plasmid design in which linkers between functional parts are re-used systematically (see Table 1-A). Yet, a newcomer in the lab would not easily be able to look at this template and build a plasmid from scratch. Lab researchers have developed their own approaches on how to efficiently build new pMOD plasmids such as reusing fragments in combination to form their plasmid backbone. Therefore, by relying on user data, it is possible to mine the information and extract local knowledge that pertains to the best practices as developed over time by the researchers in the lab.

2. METHODS

2.1 Formalization of Gibson assembled plasmids as cycled directed graphs

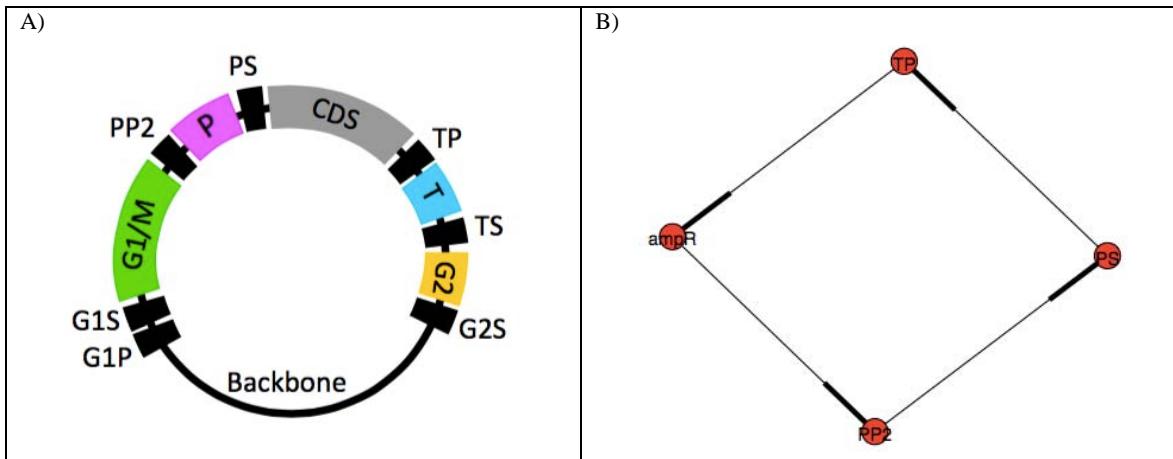
In order to understand how lab users have been building their plasmids, we mine the data associated with the Gibson assembly jobs performed in Aquarium.

We used a directed graph to formalize the way in which pMOD fragments are combined to generate a plasmid. The nodes represent the fragment overlaps and in most cases, they are pMOD linkers (as expected). The edges are fragments, directed consistently with the orientation of the gene of interest.

When reconstructing these plasmids from their list of fragments, we were often missing sequence information about the fragment. However, most of them were built in Aquarium and had their primers with sequences in the database. Using pymbt, a sequence-level DNA design specification language, it was often possible to reconstruct the overlap with the Gibson Assembly reaction module. If it fails to find a solution, we also try to automatically find identical words in the fragments' names.

If we could find a cycle in the directed graph, we had a plasmid fully reconstructed. Otherwise, we fill the gaps based on user feedback in a supervised learning fashion in order to finalize the reconstruction of the plasmid. For instance, an expert synthetic biologist may teach the graph analyzer to always associate certain nodes based on their names, skip plasmids that were not successfully built or indicate that a fragment is encoded on the opposite strand so that the directed edge is flipped.

Table 1: A) A theoretical pMOD plasmid that integrates at native yeast markers and B) the most frequent graph mined from Aquarium’s Gibson Assembly logs has a URA|ampR backbone.



2.2 Inferring a probabilistic context-free grammar (PCFG) of our lab plasmid design methods

We generated a pMOD context-free grammar (CFG) based on the re-constructed plasmids. First, we created fragment categories based on the leftmost nodes and rightmost nodes. Essentially, these CFG are ‘flat’: there is a rule to transform plasmid into the different fragment categories. Each category is then associated to its corresponding fragments following the GenoCAD library architecture [3].

We then add some ‘depth’: we automatically identified motifs, that is, two or more categories that we find together in two or more unique rules, and created intermediate rules. For instance, the URA|ampR backbone is in practice often broken down into two fragments (TP-ampR and ampR-PP2) and the homology is located on the ampR (see Table 1-B). Hence, Apprentyeast created the rule TP-PP2 → TP-ampR, ampR-PP2. Apprentyeast learnt to split ampR, which in practice will decrease the chance of template vector making its way into the Gibson reaction.

Finally, we computed the probabilities by using relative frequency estimation since rules may appear more than once. Therefore, it becomes possible to identify the dominant Gibson assembly strategies in the lab.

3. RESULTS AND FUTURE DIRECTIONS

Apprentyeast is able to design pMOD formatted DNA fragments and primers from public databases and connect with Aquarium through an API. Apprentyeast has been learning how to build plasmids from our lab practitioner data. We successfully tested it to implement and build designs in a URA integrating plasmid: we only specified the names of the yeast genes we would like to overexpress and the Aquarium’s promoter fragment we would like to use. Apprentyeast created the new gene fragments along with the necessary pMOD compliant-primers to PCR it out of a yeast lysate and selected all the Aquarium fragments to finalize a pMOD plasmid and finally asked Aquarium to make each plasmid using a Gibson Assembly task. Apprentyeast successfully

demonstrated that it can automatically have plasmids constructed in Aquarium from high-level specifications.

We are currently integrating formal semantics to Apprentyeast: the plasmids generation will formally handle the integration markers and manage these ‘logistic’ integration aspects when constructing a synthetic yeast.

In addition, after importing all existing DNA fragments found in Aquarium, we are theoretically able to generate all pMODs that could be build straightforwardly in Aquarium. Because it is possible to associate the generation of the constructs’ equations to grammars [4], we will be able to perform design-space exploration based on mass-action equations for each plasmid. We will also make it possible to generate plasmids based on an equation set.

4. ACKNOWLEDGMENTS

This work was supported by NSF Grant No. 1317653.

5. REFERENCES

- [1] E. C. Hayden, “The automad lab,” *Nature*, vol. 516, pp. 5–6, 2014.
- [2] E. Klavins, “The Aquarium Project,” 2015. (klavinslab.org/aquarium.html)
- [3] M. L. Wilson, S. Okumoto, L. Adam, and J. Peccoud, “Development of a domain-specific genetic language to design *Chlamydomonas reinhardtii* expression vectors.,” *Bioinformatics*, pp. 1–7, Nov. 2013.
- [4] Y. Cai, M. W. Lux, L. Adam, and J. Peccoud, “Modeling structure-function relationships in synthetic DNA sequences using attribute grammars.,” *PLoS Comput. Biol.*, vol. 5, no. 10, p. e1000529, Oct. 2009.

Big Mechanism Design and Analysis Automation

Anuva Kulkarni

Electrical and Computer Engineering
Carnegie Mellon University
anuvak@andrew.cmu.edu

Cheryl Telmer

Biological Sciences
Carnegie Mellon University
ctelmer@andrew.cmu.edu

Natasa Miskov-Zivanov

Electrical and Computer Engineering
Carnegie Mellon University
nmiskov@andrew.cmu.edu

ABSTRACT

In this paper, we describe a framework for automated development of executable models using information extracted from literature. The framework also includes model analysis and correction methods. The final objective is to have a representation for models of complicated mechanisms that allows for easy model exchange and improvement, and therefore facilitates the discovery of interventions (e.g., treatments or drugs in case of cell mechanisms).

1. INTRODUCTION

Understanding complicated mechanisms usually requires collecting existing information from various sources and integrating it all within a model. The most common sources include published literature, existing published models, and relevant data. By designing a single model that can be exchanged, tested and improved, we can advance and accelerate knowledge exchange in the scientific community. For a particular system, such as a biological cell signaling network, a large number of models in existing literature that were developed over the years are rendered useless when they cannot be updated, validated or corroborated with each other.

Manual processing of such tremendous amounts of data is not possible. For example, in a single model of epidermal growth factor receptor (EGFR) signaling proposed by Chen et al. in [1], there are 499 ODEs, 828 reactions and 229 parameters [2]. Hence our aim is to automate reading and model building procedures, followed by model checking and improvement. Section 2 describes the cell signaling model that we have chosen to demonstrate our method on. Figure 1 depicts our proposed framework and the various aspects of the work, with details in Section 3. With the assistance from experts in natural language processing, causal inference and cancer immunology, we hope to achieve the goal of developing a system that can learn, execute and manage models for large, complicated mechanisms, thus enabling informative simulations.

2. CASE STUDY

In our case study model, we focus on signal transduction pathways

in cancer. We would like to expand beyond the downstream effectors of Ras protein signaling to include metabolic effects and extracellular communication in the tumor microenvironment and immune system. Stress delivered to cells can result in imbalances in metabolism that may affect mitochondrial and nuclear exchange and function. Changes in metabolism are also observed when mutations in Ras occur. When the system is overwhelmed and feedback is compromised, damage control mechanisms in cells do not function properly and cells become cancerous. There are hallmark effects of stresses on nuclear chromatin structure and proteins involved with chromatin structure. The High-Mobility Group Protein B1 (HMGB1) has various cellular locations and has roles in oxidative stress, Reactive Oxygen Species (ROS) and Redox signaling. Along with these intracellular pathways, the tumor microenvironment is also an important aspect of cancer. Therefore, extracellular signaling by direct contact or through exosomes [3] needs to be considered when developing models. Immune cells are of particular interest to us and will be included in the model. Integration of models and simulations across these scales will provide greater insights and potential therapeutic targets.

3. PROPOSED FRAMEWORK

Here, we outline and explain each of the blocks in Figure 1.

3.1 Information extraction

Information for our model can come from various sources such as model databases, literature or data from experiments. Currently, information is extracted from literature using natural language processing algorithms and is entered into a standardized format that can be further processed by computers. For example, the sentence “*Cell activation with growth factors such as epidermal growth factor (EGF) induces Ras to move from an inactive GDP-bound state to an active GTP-bound state*” can be represented as:

```
CHANGE_STATE
{ Participant_1: GDP-bound Ras
  Participant_2: EGF
  Product: GTP-bound Ras
  Relationship effect: Activation }.
```

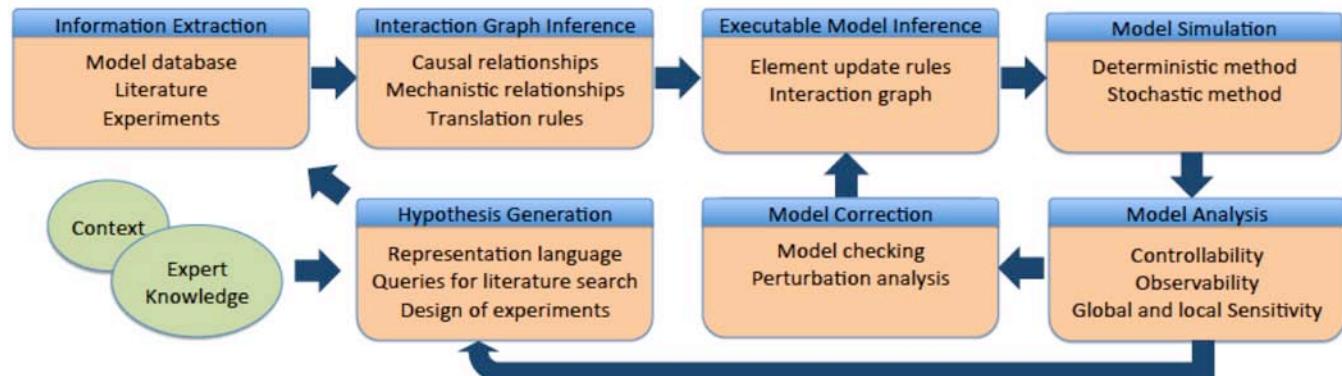


Figure 1: Our proposed framework for automated model building and analysis.

3.2 Interaction graph inference

Using the format from the previous step and causal inference algorithms, one can obtain a set of nodes and edges for the entities in the model. A visual representation of one connection inferred from our example is as seen in Figure 2.



Figure 2: Nodes and edges are inferred based on the pathway relations obtained from frames.

Models from various model databases such as BioPAX [4], Biological Expression Language (BEL) or Pathway Logic [5], which use different representation formalisms are being translated into a unique format in order to generate an interaction graph. This is a challenging task since each representation has different levels of detail and quantification; for example, BEL is a set of discrete statements, one for each reaction while BioPAX has a wide latitude of components of the pathway and details on how pathways interact.

3.3 Executable model inference

This step involves automated inference of element update functions by combining qualitative interaction graphs and any other available quantitative information. We must identify the list of components in the graph, the list of regulators for each component and the type of regulation, as well as any interactions between regulators. In our example, assuming that no other entity influences the example reaction, the simplest RAS_GTP update rule is:

$$\text{RAS_GTP} = \text{RAS_GDP} \text{ and } \text{EGF}$$

If additional information about levels of activity of RAS_GTP, RAS_GDP and EGF is available, the rule becomes more complex, and thus, model design is only feasible with automation [6]. Furthermore, due to complex interaction networks and feedback loops, creating rules for inferring accurate models from available information becomes a challenging task.

3.4 Model simulation

We focus on two types of model simulations. The first type is a deterministic approach, which gives us the steady states in the model, and the second one is a stochastic approach which allows us to analyze transient behavior. Tools described in [6] and [7] are capable of performing both types of simulation.

3.5 Model analysis

The output data from simulations can be processed using algorithms such as Principal Component Analysis to identify key regulators (for example, elements critical in studied cell behavior) or correlations and trends in the behavior of system elements or in the overall system. Additionally, we have developed a method to conduct sensitivity analysis to identify best targets for treatment. To this end, controllability analysis can also contribute to finding specific inputs that control intermediate values or outputs.

3.6 Model correction

This step involves probabilistic and statistical model checking for both, deterministic and stochastic simulations by using and expanding the tool described in [8]. Perturbation analysis considers the effects of certain altered causal relationships in both steady-state and transient behavior [9], [10].

3.7 Hypothesis generation and testing

We are currently developing a method to automate extraction of hypotheses that will follow model simulation and analysis. This step in our framework will pose new questions leading to design of new wet lab experiments, propose refinements to the model, and guide a new literature search for information extraction to validate the generated hypotheses.

4. CONCLUSION

The focus of our project is development of a completely automated model design and analysis procedure. Many tasks out of the ones mentioned in Section 3, such as extraction of frames from literature, causal inference and model simulations have shown promising results. We are applying this approach on models of cell signaling and metabolism networks as well as on cell-cell communication. Questions in cancer immunology concerning the effect of mutant Ras on metabolism of healthy cells, effect of exosomes on immune cells and their function are of special interest to us. The results from simulations of the case study model described in Section 2 will be used to find answers to these important questions.

5. ACKNOWLEDGMENTS

This work is supported in part by DARPA award W911NF-14-1-0422. The authors would like to thank and acknowledge their collaborators on this project: Michael Lotze, Christof Kaltenmeier, Peter Spirtes and Jelena Kovačević.

6. REFERENCES

- [1] Chen WW, Schoeberl B, Jasper PJ, et al. "Input–output behavior of ErbB signaling pathways as revealed by a mass action model trained against dynamic data." *Molecular Systems Biology*. 2009;5:239. doi:10.1038/msb.2008.74.
- [2] Hlavacek WS. "How to deal with large models?" *Molecular Systems Biology*. 2009;5:240. doi:10.1038/msb.2008.80.
- [3] Mittelbrunn, María, et al. "Unidirectional transfer of microRNA-loaded exosomes from T cells to antigen-presenting cells." *Nature communications* 2 (2011): 282.
- [4] Demir E, Cary MP, Paley S, et al. "BioPAX – A community standard for pathway data sharing." *Nature biotechnology*. 2010;28(9):935-942. doi:10.1038/nbt.1666.
- [5] Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln, Carolyn Talcott, "Pathway Logic: Executable Models of Biological Networks", Electronic Notes in Theoretical Computer Science, Volume 71, April 2004, Pages 144-161, ISSN 1571-0661, [http://dx.doi.org/10.1016/S1571-0661\(05\)82533-2](http://dx.doi.org/10.1016/S1571-0661(05)82533-2).
- [6] N. Miskov-Zivanov, D. Marculescu, and J. R. Faeder, "Dynamic behavior of cell signaling networks: model design and analysis automation." in Proc. of Design Automation Conference (DAC), Article 8, 6 p., June 2013.
- [7] N. Miskov-Zivanov, P. Wei, C.S.C. Loh, "THiMED: Time in Hierarchical Model Extraction and Design," in Proc. Of Computational Methods in Systems Biology (CMSB), pp. 260-263, November 2014.
- [8] N. Miskov-Zivanov, P. Zuliani, E. M. Clarke, and J. R. Faeder, "Studies of biological networks with statistical model checking: application to immune system cells," in Proc. Of ACM Conference on Bioinformatics, Computational Biology and Biomedicine (ACM-BCB), September 2013, pp. 728.
- [9] A. Garg, K. Mohanram, A. Di Cara, G. De Micheli, and I. Xenarios, "Modeling stochasticity and robustness in gene regulatory networks", Bioinformatics, vol. 25, pp. i101-i109, 2009.Tavel, P. 2007. *Modeling and Simulation Design*. AK Peters Ltd., Natick, MA.
- [10] N. Miskov-Zivanov, M. S. Turner, L. P. Kane, P. A. Morel, and J. R. Faeder, "The duration of T cell stimulation is a critical determinant of cell fate and plasticity," in Science Signaling, 6, ra97, November 2013.

On the complexity of codon context optimization

Dimitris Papamichail

Department of Computer Science

The College of New Jersey

2000 Pennington Rd, Ewing, NJ, USA

+1-609-771-2268

papamicd@tcnj.edu

Hongmei Liu

Division of Biostatistics

Department of Public Health Science

Miller School of Medicine

University of Miami, Miami, FL, USA

h.liu7@med.miami.edu

Georgios Papamichail

Department of Informatics

New York College

Athens, Greece

+30-210-349-6211

pmichael@ekdd.gr

ABSTRACT

Codon context has been shown to affect mRNA translational efficiency, but existing methods and tools for designing codon context optimized synthetic genes do not provide quantifiable measures for evaluating design quality. In this study we examine statistical properties of codon context measures and algorithms for codon context optimization under reasonable constraint models.

Categories and Subject Descriptors

F.2.0 [Analysis of Algorithms and Problem Complexity]: General.

General Terms

Algorithms, Performance.

Keywords

Codon context, gene design.

1. INTRODUCTION

Expression of genes is fundamental to modern biotechnology. Several steps in the gene expression process may be modulated, including transcription, RNA splicing, translation and post-translational modification of a protein. We will primarily focus on the process of translation, and the effect that synonymous mutations in a protein-coding gene confer to the expression of the corresponding protein. Working towards the objectives of synthetic biology, precise protein expression control has direct implications in improving heterologous expression, and in successfully designing and fine-tuning gene regulatory networks.

In most species, synonymous codons are used at unequal frequencies. Codon usage bias is recognized as crucial in shaping gene expression and cellular function, affecting diverse processes from RNA processing to protein translation and protein folding. Rarely used codons have been associated with rare tRNAs and have been shown to inhibit protein translation, where favorable codons have the opposite effect, something that is particularly pronounced in prokaryotic organisms [11]. The use of particular codons through synonymous mutations has been shown in certain cases to increase the expression of transgenes by more than 1000-fold [7].

Gutman and Hatfield first noticed that codon pairs in prokaryotic

genes exhibit another significant bias towards specific combinations [8], where codon pair optimization influences translational elongation step times [10]. More recent work by [3], [13], and [4] who synthesized novel coding regions utilizing large scale codon pair optimization and de-optimization, coupled with de novo synthesis of the constructs and in-vivo experimentation, provided further evidence of the influence codon pair bias has on translational efficiency.

Several mathematical methods have been proposed for the study of codon context bias, including [1, 3, 5, 9, 12, 14]. Several published gene design tools provide functionality for controlling codon context, albeit no two tools share the same measure of codon context bias. *Eugene* [6] is a standalone tool developed for multi-objective gene optimization and provides functionality to optimize mRNA codon context bias, but uses ‘percentages’ to indicate improvement towards a target objective instead of scores, precluding quantification of results and comparison to other methods. *Codon Optimization OnLine (COOL)* [2] is a web-based utility that can optimize for multiple objectives including codon context bias. The optimization process uses a genetic algorithm to produce several approximately pareto-optimal solutions given a set of design criteria. Codon context optimization is based on matching a given host codon pair distribution and no cumulative score is available to quantify the end result.

All current methods and tools have severe limitations, the most crucial being a lack of reference information about the optimization objectives and the optimality of the designs. Arbitrary scores are used to quantify codon and codon context bias, and it is hard to justify the use of one method over another. In this paper we focus on codon context and attempt to shed light on its statistical properties, as well as exact and approximate methods to evaluate the quality of an optimized design.

2. CODON CONTEXT

2.1 Statistical properties of codon context bias

A measure of codon context bias can be defined from the following formula of codon pair score (CPS) [3] (supporting online material, Fig.S1):

$$CPS_{AB} = \log \frac{O_{AB}}{E_{AB}} = \log \frac{F(AB)}{\frac{F(A) * F(B)}{F(X) * F(Y)} * F(XY)}$$

where the codon pair AB encodes amino acid pair XY , F denotes the number of occurrences, O is the observed number of occurrences and E is the expected number of occurrences.

Codon pair bias (CPB) for an entire gene sequence is the arithmetic mean of codon pair scores for all pairs making up the entire gene sequence.

$$CPB_k = \sum_{i=1}^k \frac{CPS_i}{k}$$

where $k+1$ is number of codons in the gene sequence.

We have shown that under the above definitions the CPS measure is independent of codon bias, scores of different codon pairs are mutually independent, and the CPB values of protein variants of a given gene are approximately normally distributed (proofs omitted). As such, the mean, standard deviation, and variance of such scores can be approximated, and the p-value of a protein being encoded by an mRNA with a specific CPB can be calculated. This information can be used to determine the significance of a specific codon bias score of an mRNA and used to estimate the effect on the gene's expression levels.

Despite differing definitions of alternative codon context bias measures, the statistical properties and computational methods discussed in this paper are applicable to all measures where codon context is evaluated in regions of constant size surrounding a codon, a property common among all examined measures in current literature.

2.2 Codon context optimization

We have proved that it is possible to efficiently design an mRNA encoding of a given protein with optimal CPB, maximized or minimized, using dynamic programming. Our algorithm involves a linear scan of the amino acids of the protein in order, keeping track of the best score thus far for each codon at the currently examined position. CPS depends only on the codon pair examined, meaning preceding and following selections of synonymous codons do not affect the score of the currently considered codon pair. Although we need to consider all codon pairs corresponding to each amino acid pair, we need only retain the maximal (or minimal) score for each codon pair chain ending at the currently examined codon, together with a pointer to the 'parent' codon that ended that chain before the current position. The pointer information can be used to retrieve the mRNA encoding with the maximal (minimal) bias once the optimal score is calculated and optimal chain identified. This process is analogous to the calculation of the edit distance of two strings and the subsequent generation of their alignment.

Codon bias is one of the primary design objectives aiming to control gene expression, and commonly it is desirable to control the codon distribution while optimizing the CPB of a protein mRNA encoding. Codon pair bias optimization while maintaining a fixed codon distribution is a hard computational problem and we have characterized it as a variant of the Travelling Salesman Problem (TSP), albeit one with polynomial time complexity as a function of the protein size [13]. We designed a dynamic programming algorithm to maximize/minimize CPB under codon distribution restrictions, which keeps track of every possible assignment of available codons to previously encountered amino acids and their optimal scores at each step. This algorithm has $O(n^{65})$ time complexity (for a protein with n amino acids), which we can reduce to $O(n^{42})$ by taking advantage of the interdependence of synonymous codons, meaning their degrees of freedom (as, for example, one codon frequency is sufficient to determine the frequency of the single alternate synonymous codon for a given number of occurrences of their corresponding amino acid).

Due to impracticality of the aforementioned algorithm, we designed and implemented a branch and bound algorithm to optimize CPB of a protein encoding under a fixed codon distribution, but, despite significant speedup, this solution is still not practical for any but the smallest proteins (<100 amino acids).

Practical approximations of optimal codon pair bias designs under a fixed codon distribution involve the use of metaheuristics such as simulated annealing, a technique particularly appropriate for TSP problem variants. We have successfully used simulated annealing to approximate optimal CPB designs within a 99/100 ratio for small proteins (<100 amino acids).

Software implementations of all aforementioned algorithms are available from the authors upon request. A web-based utility is currently under construction.

3. ACKNOWLEDGMENTS

This work has been supported by NSF Grant CCF-1418874.

4. REFERENCES

- [1] Boycheva, S., Chkodrov, G. and Ivanov, I. 2003. Codon pairs in the genome of Escherichia coli. *Bioinformatics*. 19, (2003), 987–998.
- [2] Chin, J.X., Chung, B.K.-S. and Lee, D.-Y. 2014. Codon Optimization On-Line (COOL): a web-based multi-objective optimization platform for synthetic gene design. *Bioinformatics (Oxford, England)*. (2014), btu192.
- [3] Coleman, J.R., Papamichail, D., Skiena, S., Futcher, B., Wimmer, E. and Mueller, S. 2008. Virus attenuation by genome-scale changes in codon pair bias. *Science*. 320, 5884 (2008), 1784–1787.
- [4] Coleman, J.R., Papamichail, D., Yano, M., Garcia-Suarez Mdel, M. and Pirofski, L.A. 2011. Designed reduction of Streptococcus pneumoniae pathogenicity via synthetic changes in virulence factor codon-pair bias. *J Infect Dis*. 203, 9 (2011), 1264–1273.
- [5] Fedorov, A., Saxonov, S. and Gilbert, W. 2002. Regularities of context-dependent codon bias in eukaryotic genes. *Nucleic acids research*. 30, (2002), 1192–1197.
- [6] Gaspar, P., Oliveira, J.L., Frommlet, J., Santos, M.A.S. and Moura, G. 2012. EuGene: Maximizing synthetic gene design for heterologous expression. *Bioinformatics*. 28, (2012), 2683–2684.
- [7] Gustafsson, C., Govindarajan, S. and Minshull, J. 2004. Codon bias and heterologous protein expression. *Trends Biotechnol*. 22, 7 (2004), 346–353.
- [8] Gutman, G.A. and Hatfield, G.W. 1989. Nonrandom utilization of codon pairs in Escherichia coli. *Proc Natl Acad Sci U S A*. 86, 10 (1989), 3699–3703.
- [9] Hooper, S.D. and Berg, O.G. 2002. Detection of genes with atypical nucleotide sequence in microbial genomes. *Journal of molecular evolution*. 54, (2002), 365–375.
- [10] Irwin, B., Heck, J.D. and Hatfield, G.W. 1995. Codon pair utilization biases influence translational elongation step times. *J Biol Chem*. 270, 39 (1995), 22801–22806.
- [11] Lithwick, G. and Margalit, H. 2003. Hierarchy of sequence-dependent features associated with prokaryotic translation. *Genome Res*. 13, 12 (2003), 2665–2673.
- [12] Moura, G., Pinheiro, M., Silva, R., Miranda, I., Afreixo, V., Dias, G., Freitas, A., Oliveira, J.L. and Santos, M.A.S. 2005. Comparative context analysis of codon pairs on an ORFeome scale. *Genome biology*. 6, (2005), R28.
- [13] Mueller, S., Coleman, J.R., Papamichail, D., Ward, C.B., Nimmulal, A., Futcher, B., Skiena, S. and Wimmer, E. 2010. Live attenuated influenza virus vaccines by computer-aided rational design. *Nat Biotechnol*. 28, 7 (2010), 723–726.
- [14] Shah, A.A., Giddings, M.C., Parvaz, J.B., Gesteland, R.F., Atkins, J.F. and Ivanov, I.P. 2002. Computational identification of putative programmed translational frameshift sites. *Bioinformatics (Oxford, England)*. 18, (2002), 1046–1053.

Context-Aware Pipetting

Charles Fracchia^{*}
MIT Media Lab
20 Ames Street
Cambridge, MA 02139
fracchia@mit.edu

Prof. George Church
Harvard Medical School
77 Avenue Louis Pasteur
Boston, MA 02115
gmc@harvard.edu

Prof. Joseph Jacobson
MIT Media Lab
20 Ames Street
Cambridge, MA 02139
jacobson@media.mit.edu

ABSTRACT

Pipetting operations are at the core of nearly every molecular biology protocol. They provide a way to precisely move, mix or separate liquids, whether they be samples, reagents or other materials. However, a major issue of manual pipetting rests in the feedback it provides to the user. Further issues arise when pipetting small volumes or when performing a large number of operations due to the cognitive burden associated with remembering accurately every action. This leads to uncertainty of sample content, wasted reagents and contributes to making biological research difficult and expensive.

While recent efforts[9] have demonstrated the use of a web application to provide visual pipetting instructions for well plates, the system does not incorporate feedback of the user's action. Other systems[3][2][1] rely on custom hardware to provide the pipetting instructions but still lack the ability to have real-time feedback, on the user's actions.

To solve this problem, we have developed a computer vision system able to track the position of the pipette tip relative to a plate in real-time. We also show the reverse-engineering of an electronic pipette that allows to operate the system in a closed-loop and drive the actuation of the pipetting operation automatically. Due to its simple hardware requirements, this system is readily retrofitted to existing lab furniture, providing pipette tracking capabilities with minimal workflow adjustment on the part of the user. A video of the pipetting tracking system is available at:
<http://vimeo.com/charlesfracchia/welltracker>

Keywords

pipetting, computer vision, reverse engineering

1. SYSTEM ARCHITECTURE

The different components of the context-aware pipetting system are outlined in the system architecture in Figure 2.

The system is composed of two main parts: the computer vision system and the electronic pipette. We have built the system to relax as many constraints as was possible and reasonable. Currently, the system only requires the use of a USB camera overhead and special tips whose sole modifications are their colored ends. The color can be changed in the code and is therefore not restricted to our arbitrary choice of blue.

This project makes use of the popular OpenCV[4] library to carry out the computer vision portion of this work. The

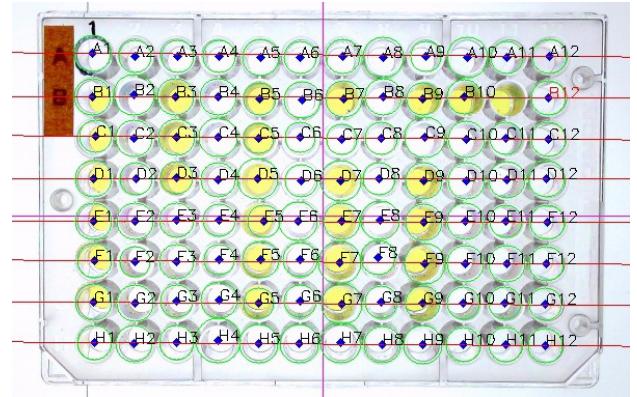


Figure 1: Annotated view of the plate tracking system. In yellow: the wells that the user pipetted into

task of tracking the user's pipetting patterns can be subdivided into two problems: first, obtaining positions and naming each of the wells in the plate being used and, second, obtaining the current position of the pipette tip. Once these two subproblems are solved, the current well into which the user is pipetting can be extracted by comparing the position of the tip to the position of the closest circle.

2. WELL AND TIP DETECTION

The computer vision task of tracking and annotating well plates rests on the assumption that a defining, recognizable pattern can be extracted and tracked in a sufficiently agnostic way as to avoid restricting the user in their choice of plate. The number of wells in a plate is determined by the number of rows and columns regimenting the grid pattern in which the wells are arranged. Plates are often referred to by the number of wells they have. Common formats are either 96-well plates: 8 rows by 12 columns; and 384-well plates: 16 rows by 24 columns.

With the exception of some cell growth plates, the vast majority of plates have circular wells arranged in the grid pattern previously described. We therefore decided to base our detection on this common feature. Circle detection is performed in our system by using the OpenCV HoughCircles[5] transform. Internally, this transform first performs edge detection using the Canny algorithm[7] before using a gradient search method[8] to detect circles on the image. The algorithms using the transform each have input parameters that affect the overall result of the circle detection.

*Corresponding Author

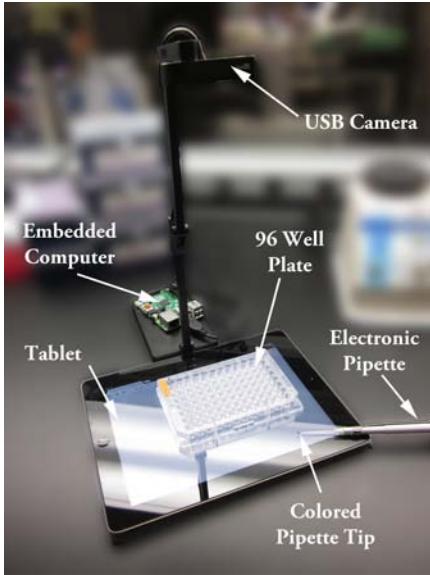


Figure 2: The overhead camera provides video to the embedded computer running the computer vision code in real-time. The code detects the plate, annotates its wells and sends a message to the Web interface displayed on the tablet to provide real-time pipetting feedback to the user. The tablet is solely used to provide localised, intuitive feedback to the user and is not a required component of the system. An electronic pipette is used to *close the loop* in our system and prevent pipetting into the wrong well.

The parameters were chosen to ensure maximum detection of the circles representing the wells in varied lighting and viewing angle conditions.

Key to determining the bounds of a well by solely using the circular well as the feature is the ability to detect the edge wells, in particular the four corner wells. In order to reduce search time through all detected circles from the previous step, the image is split into four equally sized quadrants. The corner wells are then detected by computing the distance from each of the wells within the quadrant to their respective corners.

Using the center of each well obtained from the circle detection step described previously, we fit a line passing through both corner wells. Using the knowledge that wells are arranged following a grid pattern, we thus expect all *A* row wells to be within some minimal distance from the fitted line. Once all row wells are detected, we iterate from both edge wells at columns 1 and 12 inward to associate the detected circle with the particular well.

In order to mitigate the effect of lighting artifacts, we check the distance between adjacent wells and if it is found to be larger than the average distance between detected circles on the overall plate, we instantiate the missing circle along the fitted line. This allows us to fill in wells that may be missing due to lighting or other aberrations. The whole process is then repeated for each row. The following row edge wells are determined by calculating the wells with minimum distance from the previous edge wells but that are not along the previously fitted row axis. The end result of this row-by-row annotation is shown in Figure 1. Tests con-

ducted in different lighting conditions and plate orientations show that this annotation algorithm is robust.

In order to successfully and reliably track the position of the tip, we opted for tracking a custom-colored tip by applying color filtering to the frame. The choice of color is arbitrary and can be readily changed.

3. REVERSE ENGINEERING THE EDP3

In order to demonstrate closed-loop pipetting operations, we reverse engineered an electronically drivable pipette. This serves as a proof of concept application for incorporating actuation in the loop, allowing to assist pipetting operations when the user is over the correct well. Using a logic analyzer[6] wired to each of the terminals on the battery connector, we were able to collect all communications from the EDP3 electronic pipette. However, in its normal state, the pipette did not seem to output any data. Thanks to the help of a Rainin engineer, we were able to obtain a compiled version of the Windows utility used to control the EDP3 remotely. Using a USB FTDI Serial cable and the logic analyzer, we captured all available commands and their responses by the pipette. We then created a custom board that plugs in the back port and enables wireless control (802.15.4) of the pipette. The schematics and firmware for the board, as well as the communication commands for the pipette are available by contacting the author.

4. CONCLUSIONS

This work describes the creation of a pipetting system that is closed loop and context aware. While the current incarnation of the system relies on the presence of a blue marker at the extremity of the tip, further work on classifiers might enable its operation without any special tips. The work outlined in this paper enables the future insertion of controls systems, thus carrying the promise of increased traceability, reproducibility and reliability of this key biological lab activity.

5. REFERENCES

- [1] ACTGene SpotLITER 96well Plate Light Tracker. <http://www.actgene.com/SpotLITER.html>.
- [2] Embi Tec LI-2100 LightOne Pro. <http://embitec.com/li2100-lightone-pro-384-and-96-well.html>.
- [3] Gilson TRACKMAN. <http://www.gilson.com/en/Pipette/Products/45.265/Default.aspx>.
- [4] Open Computer Vision package. <http://opencv.org/>.
- [5] OpenCV Hough Circles Function Documentation. http://docs.opencv.org/modules/imgproc/doc/feature_detection.html?highlight=houghcircles#houghcircles.
- [6] Saleae Logic16 Logic Analyzer. <https://www.saleae.com/logic16>.
- [7] J[ohn] Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, June 1986.
- [8] H. Yuen, J. Princen, J. Illingworth, and J. Kittler. A comparative study of hough transform methods for circle finding. In *Proc. 5th Alvey Vision Conf.*, pages 169–174, Aug. 1989.
- [9] D. Zielinski, A[ssaf] Gordon, B[enjamin] L Zaks, and Y[aniv] Erlich. iPipet: sample handling using a tablet. *Nature Methods*, 11:784–785, July 2014.

CRISPR and TAL Search and Design Tools for Cell Engineering

Daniel Williams
Thermo Fisher Scientific
5791 Van Allen Way
Carlsbad, CA 92008, USA
1-805-637-0856
dw314159@gmail.com

Sridhar Ranganathan
Thermo Fisher Scientific
5791 Van Allen Way
Carlsbad, CA 92008, USA
1-760-268-5360
Sridhar.Ranganathan@lifetech.com

Joel Brockman
Thermo Fisher Scientific
2130 Woodward Street
Austin, TX 78744, USA
1-512-721-3859
Joel.Brockman@lifetech.com

ABSTRACT

CRISPRs, TALs, and siRNAs are efficient tools for engineering cell activity. CRISPRs and TALs in particular edit the genome, allowing knock out or knock in of functionality. This makes them prime resources in the synthetic biologist's toolkit. We provide a web-based computational platform for selecting CRISPR and TAL designs out of a database of designs targeting most human and mouse genes. Furthermore, we provide computational tools for the *de novo* generation of designs to arbitrary target sequences.

General Terms

Algorithms, Design, Standardization

Keywords

Cell engineering, synthetic biology, CRISPR, TAL, design

1. INTRODUCTION

CRISPRs and TALs enable efficient knock out and knock in of functionality in a cell, making them powerful engineering tools. While online design algorithms exist, there is currently no standardized pool of CRISPR and TAL designs. To remedy this situation, we created databases of CRISPR and TAL designs produced using our design algorithms targeting most human and mouse genes. Furthermore, we have made these databases accessible through a web interface. For cases where a predesigned CRISPR or TAL does not exist, we provide through the same web interface direct access to our design algorithms, enabling users to enter arbitrary target sequences and retrieve suitable designs.

2. METHOD

2.1 Design Algorithms

Our CRISPR and TAL design algorithms implement design rules derived from analysis of laboratory data and best practices [1, 2, 3]. For a given target, a pool of potential candidate CRISPRs or TALs is generated and each is scored by one of the algorithms. The top-scoring candidates are then selected. The scoring strategy balances predicted effectiveness of a CRISPR or TAL versus its predicted risk of off-target effects.

2.2 Databases

The CRISPR and TAL MySQL design databases reside in the cloud, hosted by Amazon's Relational Database Service.

2.3 APIs

Web-based APIs connect the databases and design algorithms to the user interface, providing responses in JSON format easily processed by the user interface. The APIs are written in Java and are hosted on Amazon EC2 instances.

2.4 Web Interface

The web interface providing access to the CRISPR and TAL design databases and to the design algorithms is written in JavaScript and hosted on an Amazon E2 instance. Figure 1 shows the CRISPR search page. Figure 2 shows corresponding search results.

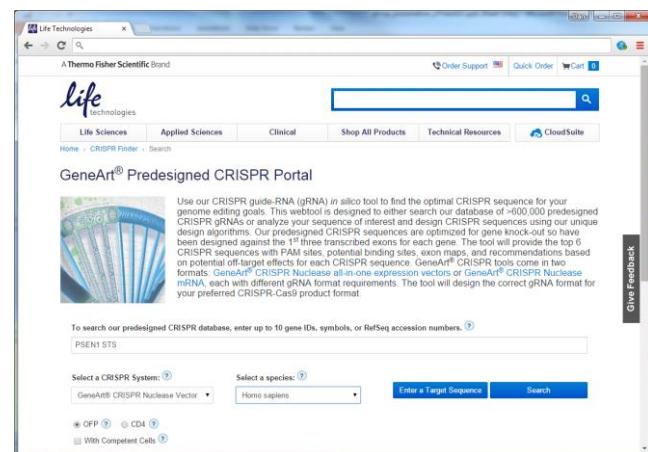


Figure 1. CRISPR Search Interface

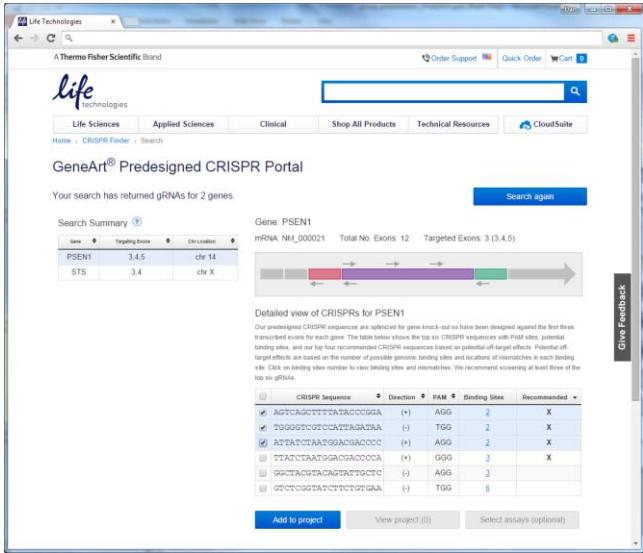


Figure 2. CRISPR Search Results

3. RESULTS

3.1 Gene Coverage

Our CRISPR, TAL, and siRNA designs target a total of 18,002 genes, where each technology targets each gene. This enables researchers to choose more than one method for the same gene. Furthermore, our CRISPR designs target an additional 148 genes, our TAL designs target an additional 2,559 genes, and our siRNA designs target an additional 241 genes. This overlap in targeting is shown in the Venn diagram in Figure 3.

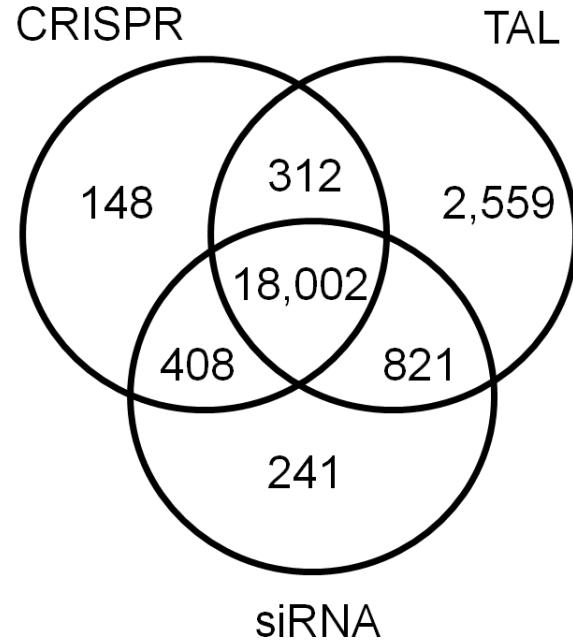


Figure 3. Gene Coverage

4. ACKNOWLEDGMENTS

Our thanks to Kevin Clancy for leadership during the development of these projects.

5. REFERENCES

- [1] Doench, J., Hartenian, E., Graham, D., Tothova, Z., Hegde, M., Smith, I., Sullender, M., Ebert, B., Xavier, R., and Root, D. 2014. Rational design of highly active sgRNAs for CRISPR-Cas9-mediated gene inactivation. *Nature Biotechnology*. 32 (Sept. 2014), 1262-1267. DOI=10.1038/nbt.3026.
- [2] Tsai, S., Zheng, Z., Nguyen, N., Liebers, M., Topkar, V., Thapar, V., Wyveldens, N., Khayter, C., Iafrate, A., Le, L., Aryee, M., and Joung, K. 2015. GUIDE-seq enables genome-wide profiling of off-target cleavage by CRISPR-Cas nucleases. *Nature Biotechnology*. 33 (2015), 187-197. DOI=10.1038/nbt.3117.
- [3] Cermak, T., Doyle, E., Christian, M., Wang, L., Zhang, Y., Schmidt, C., Boller, J., Somia, N., Bogdanove, A., Voytas, D. 2011. Efficient design and assembly of custom TALEN and other TAL effector-based constructs for DNA targeting. *Nucleic Acids Research*. 39 (July 2011). DOI=10.1093/nar/gkr218.

Design and Characterization of Genetic Circuits using Multiplex DNA Synthesis

Daniel B. Goodman
Bioinformatics and Integrative Genomics Harvard-MIT Health Sciences and Technology Cambridge, MA
dbg@mit.edu

Casper Enghuus
Department of Systems Biology
Denmark Technical University Lyngby, Denmark
casperenghuus@gmail.com

George M. Church
Department of Genetics
Harvard Medical School Boston, MA
gchurch@genetics.med.harvard.edu

ABSTRACT

Oligonucleotide library synthesis can generate many thousands of high-quality DNA sequences at low cost. Coupled to reporter assays that utilize high-throughput sequencing, this platform can be used to simultaneously measure thousands of computationally designed genetic elements in a single experiment. We have developed a software framework called Promuter that can design and interpret these massively multiplex experiments. Promuter uses position weight matrices and an iterative mutational strategy to identify, score, and modify cis-regulatory elements in *E. coli* that control transcription, translation, and inducible expression. We used Promuter to generate a library of over 140,000 simple transcriptional circuits in *E. coli* and measured their input/output relationships. These circuits utilize over a dozen different synthetic and natural transcription factors, and can be used to optimize small-molecule biosensors and quantitatively tune synthetic genetic systems.

1. BACKGROUND

Gene expression is choreographed by short stretches of DNA called genetic regulatory elements. By interacting with proteins and RNAs, these elements form additional layers of genetic instructions interspersed between and within the protein-coding genes themselves. Unlike the straightforward relationship between the sequence of a gene and its corresponding protein, the relationships between the location, sequence, and function of regulatory elements are extremely complex and poorly understood. A better understanding of these relationships will have vast implications for medicine and biotechnology, such as predicting disease from natural human genetic variation and for programming synthetic genetic circuits for environmental sensing, bioproduction, and molecular diagnostics.

Our knowledge about how genetic elements function comes from comparing and measuring the activity of natural sequences, or from genetically perturbing natural elements at relatively small scales. We still have only limited ability to predict how changing these DNA elements will affect expression, and have even less understanding of how to design and tune novel regulatory systems for biotechnological applications.

2. PREVIOUS WORK

Advances in oligonucleotide synthesis now allow for the production of large amounts of inexpensive, high-quality syn-

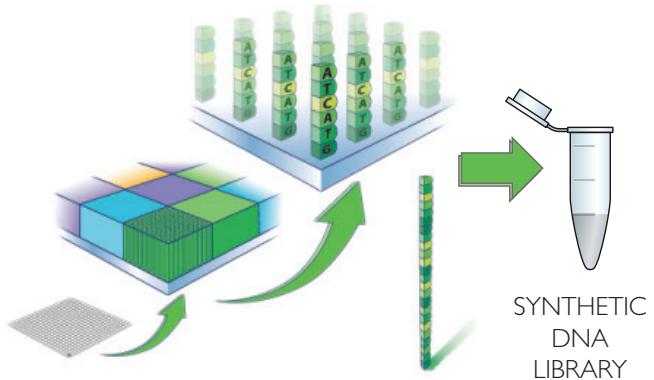


Figure 1: Up to 500,000 distinct DNA oligonucleotides, each up to 230 base-pairs long, are computationally designed and printed simultaneously onto the surface of a glass slide. After printing, the DNA is eluted, resulting in a complex library.

thetic DNA oligomers. We developed an experimental method called FlowSeq that combines oligonucleotide synthesis with fluorescence-activated cell sorting and high-throughput DNA sequencing to measure protein expression from thousands of synthetic constructs in a single experiment. Applying this approach to *E. coli*, we have previously examined the composability of regulatory elements that control transcription and translation[1], and explained why rare codons at the N-terminus of genes increase protein expression[2].

3. PROMUTER

Promuter uses strength and spacing of multiple position weight matrices to identify and score promoters and transcription factor binding sites. We use data from our previous high-throughput studies[1] to train a model of transcriptional rate based on sequence alone. To create new genetic elements, we then build a mutational landscape in which certain mutations are favored and others are disfavored, and iteratively search that landscape for combinations of mutations that satisfy constraints, such as the creation or removal of regulatory elements, or the maintenance of the current transcriptional or translational rates. We have used Promuter to generate a wide range of circuit designs by modifying number, location, and strength of binding sites, as well as circuits that self-repress and self-activate. Our circuits utilize commonly-used transcription factors like *tetR*, *lacI*,

and λcI , as well as other lambda repressors, zinc finger-based transcriptional activators, and allosteric transcription factors that sense molecules of industrial relevance.

4. FLOWSEQ

Using our FlowSeq method, we identify the input/output relationships for each circuit, identifying those that perform a variety of useful functions, such as amplification and dampening of input signals, band-pass and band-stop filters, and switch-like behavior. We also perform the experiments in media with different concentrations of small molecule inducers or at different induction time points. In addition to being useful for the programmable modulation of gene expression, a quantitative understanding of the behavior of these circuits and their constitutive elements will allow for predictive design of more complex synthetic gene regulatory systems.

5. DISCUSSION

Until now, our knowledge about how genetic elements function comes from comparing and measuring the activity of natural sequences, or from genetically perturbing natural elements at relatively small scales. The approach described here allows unlimited interrogation of sequence-space to generate thousands of simultaneous hypotheses, and can answer questions that were previously intractable. As sequencing and synthesis become cheaper, faster, and higher quality, synthetic and systems biology are exponentially accumulating predictive capacity in a rapid design-build-test cycle.

6. ACKNOWLEDGMENTS

This work was supported by the U.S. Department of Energy (DE-FG02-02ER63445) and a NSF Graduate Research Fellowship (D.B.G.).

7. REFERENCES

- [1] S. Kosuri*, D. B. Goodman*, G. Cambray, V. K. Mutalik, Y. Gao, A. P. Arkin, D. Endy, and G. M. Church, “Composability of regulatory sequences controlling transcription and translation in *Escherichia coli*,” *Proceedings of the National Academy of Sciences*, vol. 110, no. 34, pp. 14024–14029, 2013.
- [2] D. B. Goodman, G. M. Church, and S. Kosuri, “Causes and effects of n-terminal codon bias in bacterial genes,” *Science*, vol. 342, no. 6157, pp. 475–479, 2013.

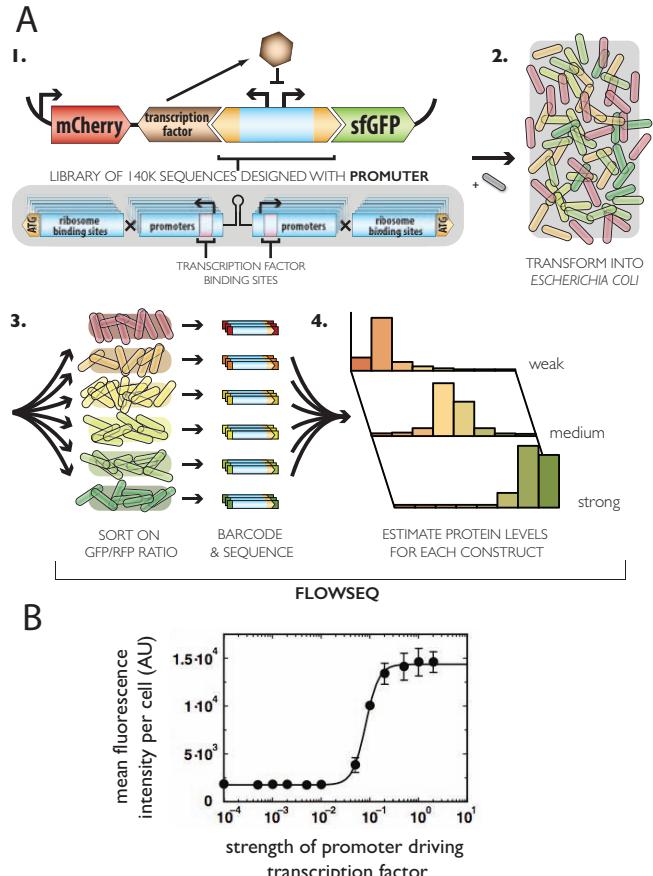


Figure 2: (A) Using Promoter, we computationally design and synthesize a library of interacting promoters and regulatory elements. The library is then cloned into a plasmid to express super-folder GFP; mCherry is independently expressed from a constitutive promoter to act as an intracellular control (1). The plasmids are transformed into *E. coli* and then FlowSeq is performed to quantify DNA and protein levels for each construct (2). In FlowSeq, cells are sorted via FACS into bins of varying GFP to mCherry ratios (3), barcoded, and sequenced with short Illumina reads to reconstruct protein levels for each individual construct (4). (B) On every oligonucleotide synthesized, the left promoter expresses a transcription factor, and a second promoter controls a reporter via a transcription factor binding site. By measuring the reporter over a range of transcription factor expression levels, we can measure a characteristic response curve for the inducible promoter.

Design optimizations of precise synthetic genome targeting and editing small molecules for diverse disease loci

Faisal Reza

Yale University

333 Cedar St., HRT 316, #208040

New Haven, CT 06520-8040, USA

+1 (347) 746-7392

faisal.reza@yale.edu

Peter M. Glazer

Yale University

15 York St., HRT 140A, #208040

New Haven, CT, 06520-8040, USA

+1 (203) 737-2788

peter.glazer@yale.edu

ABSTRACT

Triplex-forming small molecules are nanomolecular mutagens designed to precisely target a genomic locus by locally recruiting and upregulating the endogenous genome homologous recombination (HR) machinery. Upon recruitment, the HR machinery at some frequency utilizes donor DNA small molecules containing precise edits as nanomolecular recombinagens for HR-dependent genome repair safely and efficaciously. Hematopoietic progenitor cells dosed with these mutagens and recombinagens offer a regenerative treatment modality for hematological disorders, such as for a misspliced or defective adult β -globin subunit in β -thalassemia or sickle cell disease, respectively. A semi-automated rational design approach optimizes these recombinagenic and mutagenic small molecules for genome engineering and molecular therapy.

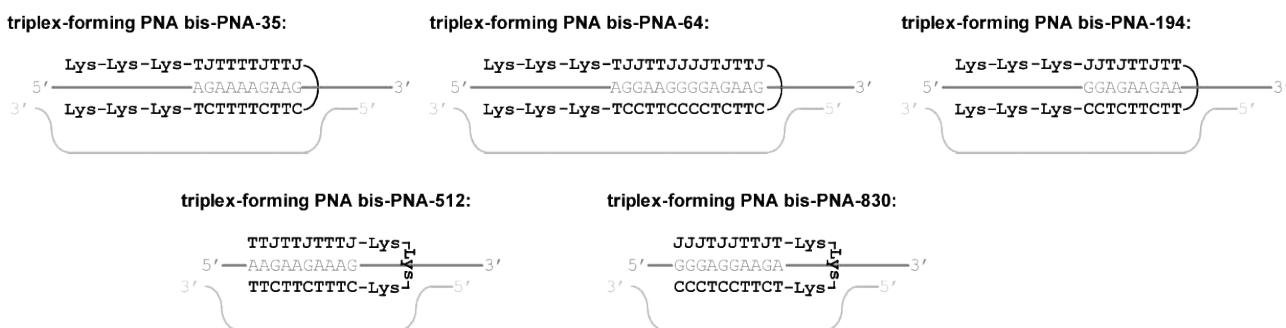
Keywords

Design, Genome engineering, Molecular therapy.

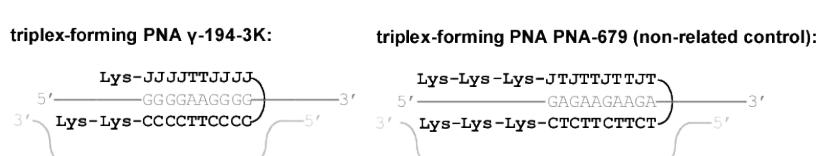
Figure 1. (A) Synthetic triplex-forming peptide nucleic acid (PNA) chemistry. **(B)** Triplex-forming PNA molecules designed as mutagens by windowing around polypurine targeting sites near human genome β -globin, γ -globin loci.

B

triplex-forming molecules for beta-globin gene targeting



triplex-forming molecules for gamma-globin gene targeting



nucleolytic degradation resistance (Fig. 2A).

2. MATERIALS AND METHODOLOGY

Mutagenic triplex-forming PNA and recombinagenic donor DNA small molecules were designed for human chromosome 11 to precisely target HR activity by windowing around polypurine targeting sites in whole human genomic DNA near the β -globin gene or γ -globin loci and verifying propensity for both Watson-Crick and Hoogsteen base pairing (Fig. 1B). The recombinagenic donor DNA designs were intended to edit the β -globin locus to repair its missplicing, or to edit the γ -globin locus to induce fetal γ -globin expression by centering and windowing off-center from the editing sites and verifying propensity for Watson-Crick base pairing (Fig. 2B).

3. RESULTS AND DISCUSSION

Designed mutagens and recombinagens precisely targeted and edited episomal and chromosomal genomic human DNA as shown by molecular weight shifts, mRNA expression, and repair frequencies consistent with repair of β -globin, and induction and regulation of γ -globin under hypoxia in mammalian progenitor cells [2].

4. CONCLUSIONS

Gene targeting mutagens and editing recombinagens designed to treat the genomic basis of β -thalassemia and sickle cell disease in regenerative progenitor cells has the potential to permanently address these hematological disorders in diverse populations [3].

5. ACKNOWLEDGMENTS

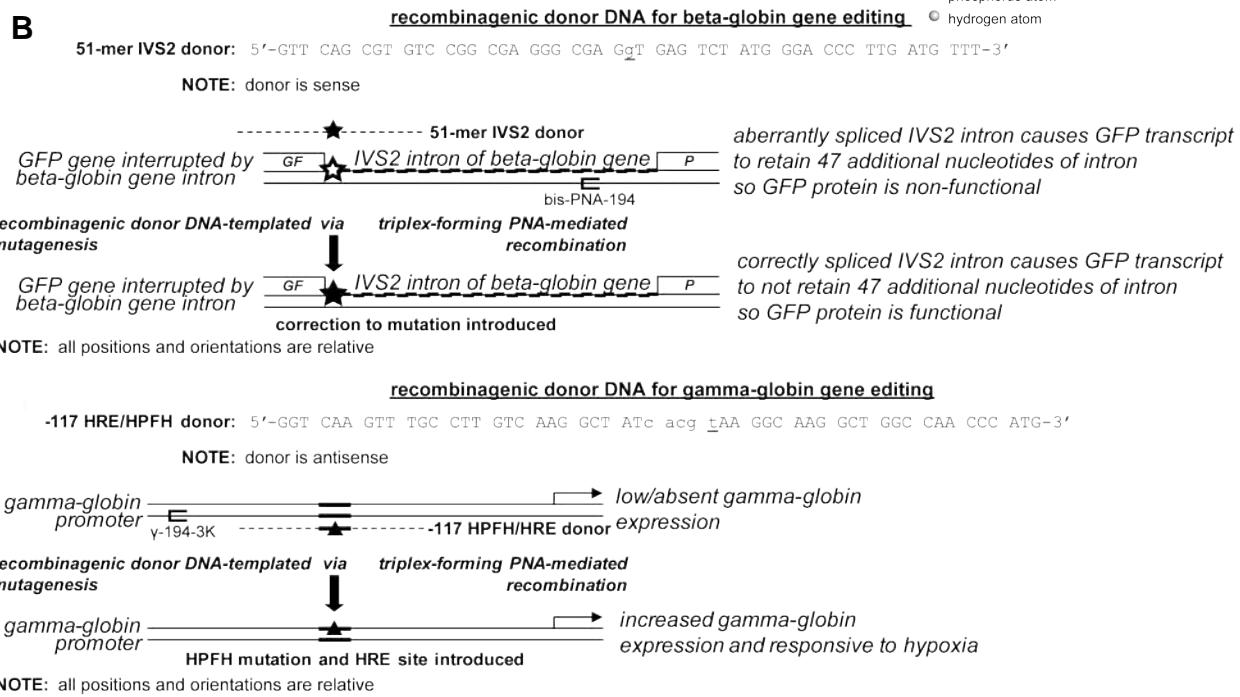
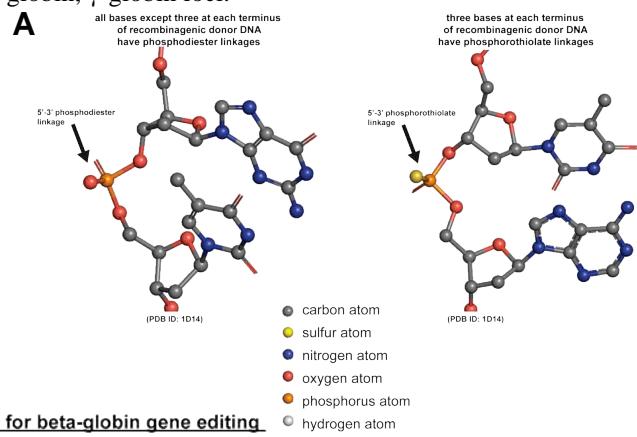
This work was supported by a National Institutes of Health (NIH) grant R01HL082655 and by a Doris Duke Innovations in Clinical Research Award (to P.M.G.). A National Institute of Diabetes and Digestive and Kidney Diseases Experimental and Human Pathobiology Postdoctoral Fellowship from NIH grant T32DK007556

also provided support (to F.R.). An International Workshop on Bio-Design Automation (IWBDA) Scholarship from the National Science Foundation (NSF) and the Bio-Design Automation Consortium (BDAC) provided further support (to F.R.). The authors declared no conflict of interest.

6. REFERENCES

- [1] F. Reza, P. M. Glazer. Triplex-mediated genome targeting and editing. *Methods in Molecular Biology: Gene Correction*, 1114:115-42, 2014.
- [2] J. Y. Chin*, F. Reza*, P. M. Glazer. (* = contributed equally). Triplex-forming peptide nucleic acids induce heritable elevations in gamma-globin expression in hematopoietic progenitor cells. *Molecular Therapy*, (3):580-7, 2013.
- [3] F. Reza, P. M. Glazer. Therapeutic genome mutagenesis using synthetic donor DNA and triplex forming molecules. *Methods in Molecular Biology: Chromosomal Mutagenesis*, 1239:39-73, 2015.

Figure 2. (A) Synthetic donor DNA chemistry. **(B)** Donor DNA molecules designed as recombinagens by centering and windowing over editing sites at human genome β -globin, γ -globin loci.



A Detailed, flexible model for sharing DNA concepts

Barbara Frewen

Zymergen

6121 Hollis St.

Emeryville, CA 94608

frewen@zymergen.com

Jed Dean

Zymergen

6121 Hollis St.

Emeryville, CA 94608

jed@zymergen.com

Aaron Kimball

Zymergen

6121 Hollis St.

Emeryville, CA 94608

aaron@zymergen.com

ABSTRACT

The process of microbe engineering at an industrial scale requires storing vast amounts of DNA sequence information. Multiple teams must be able to share design ideas, scientists must be able to communicate with production engineers, and project managers must be able to track details from the early stages of conception to the final stages of evaluation. To meet these multiple needs we have designed and are building a model for communicating DNA sequence information. We present the principles of our data model here as we believe it is broadly applicable in the synthetic biology community.

1. INTRODUCTION

Zymergen is a technology-driven company applying radical new methods to design and improve microbes by rewriting their DNA. This capability allows us to generate novel chemicals, advanced materials, and pharmaceuticals. Our approach combines biology, robotic automation, and proprietary computational and analytic methods to do this work faster and cheaper than competing technology. Currently, we are working with large industrial partners helping them improve their existing production microbes. Over time we expect to develop a large portfolio of our own products, ranging from drop-in replacements for existing chemicals to advanced materials to life-saving therapeutics.

Our technical development work is focused on establishing new molecular biological techniques, moving existing protocols to robotics workstations, and creating workflows that enable high throughput microbe engineering and evaluation. Within that work we face the challenge of communicating design concepts at the level of DNA parts, DNA assemblies, and engineered cells lines: (1) between Zymergen and partner businesses, as well as (2) between Development and Operations departments within Zymergen. In addition, we anticipate that new players will arise that will provide third-party services for building and testing various aspects of our cell engineering and evaluation pipeline.

To address these data communications issues, we have designed and are building the system described herein. Our data model builds on the concepts of the Synthetic Biology Open Language v1.1 (SBOL) [1], specifically by introducing a new entity that

defines relationships between sequences.

2. Origins

At the core of our data model is the SBOL concept of a *DNA Component*: an entity describing a region of DNA. The fundamental information is the underlying DNA sequence. Decorating that sequence are *Sequence Annotations* which are comprised of a region of nucleotides in that sequence and an orientation. One elegant feature of this model is that each Sequence Annotation can become its own DNA Component. The ability to nest DNA components within each other allows details to be captured at different levels of granularity.

Consider a plasmid to illustrate these concepts. The entire plasmid can be stored as a DNA component. Within that plasmid are features like an origin of replication, an antibiotic resistance marker, and an insert. Each feature is stored as a Sequence Annotation. The plasmid insert, in turn is its own DNA Component with Sequence Annotations like promoters, genes, and terminators.

3. DNA Specification

SBOL also includes the concept of a *Collection* as a way of grouping related DNA Components. We build on the concept of a collection with a *DNA Specification*. In addition to grouping DNA Components, the DNA Specification defines relationships among related sequences. It behaves much like a function: performing a specific operation on inputs to create outputs. Operations are performed on the underlying DNA sequence. Examples include concatenating two sequences together, selecting a region from a larger sequence, or replacing one region with another.

Returning to the above example of a plasmid, we may have a set of several related plasmids. The DNA Specification describing this set would be made up of two inputs: the empty plasmid and a set of inserts. The specification would perform a replacement operation, exchanging the insertion site in the empty plasmid with each of the insert sequences. The outputs of the specification would be the final set of related plasmids. Note that the empty plasmid in our example is a DNA Component and that the set of inserts may be another DNA Specification. Just as DNA Components can be composed of other DNA Components, DNA Specifications may also be composed of other DNA Specifications.

Operations of a DNA Specification are intentionally not related to DNA function, properties, or behavior in a cellular context. They are explicitly string-manipulation functions in order to provide a clear distinction between the description of *what* DNA sequences are being designed or constructed and *how* or *why* those

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Conference '10, Month 1–2, 2010, City, State, Country.
Copyright 2010 ACM 1-58113-000-0/00/0010 ...\$15.00.

sequences are under consideration. Our tools provide a separate layer of information (not described here) to convey, for example, construction protocols and biological function of those sequences.

4. Design/build communication

Let us illustrate how this two-part model of DNA Components and DNA Specifications facilitates one of the interactions described above. In our large-scale, automated process the teams responsible for deciding what DNA assemblies are built are different than the teams responsible for producing those assemblies. The designers can communicate their ideas via a DNA Specification. The outputs provide the exact sequences that should be produced, but they also illustrate how sequences are related which can be used to inform the methods used to assemble them.

5. Flexibility of storage and presentation

The DNA Specification provides a compact storage format for a set of similar DNA Components. Shared elements, like the empty plasmid sequence in the above examples, need only be serialized once. Further compression can be achieved by relying on external resources. For example, a URI to a public database can point to a specific genome sequence.

Given the right tools for interpreting DNA Specifications, the user can be presented with only the information of interest at the correct level of detail. A researcher performing a multiple sequence alignment might need the fine-grained detail of the exact sequence of every DNA Component in a specification whereas an operator might only need a list of parts in the freezer inventory.

6. Conclusion

Communicating DNA design ideas is central to the work we do at Zymergen. Our early implementations of the data model described here are providing a rich and flexible model for that communication.

We are keenly interested in the development of SBOL 2, a project which is under development concurrent with our work on DNA Specifications. While SBOL 2 does not contain the pure textual relationships between DNA sequences, we look forward to studying this design further as it develops, and integrating with its final results.

7. REFERENCES

- [1] Galdzicki, M., et al. 2014. The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nature Biotechnology*. 32, 545-550.

Efficient Analysis of SBML Models Using Arrays

Leandro H. Watanabe

Dept. of Electrical and Computer Engineering

l.watanabe@utah.edu

Chris J. Myers

Dept. of Electrical and Computer Engineering

myers@ece.utah.edu

1. INTRODUCTION

Standards are key to the success of systems and synthetic biology since standards give biologists the ability to share models. The leading standard representation of biological systems is the *systems biology markup language* (SBML) [3]. Unfortunately, not all modeling efforts in systems biology use SBML. For example, Karr et al. [4] developed a computational model of a whole-cell using MATLAB. This model includes all of the molecular components of the cell and their interactions, which makes the model quite involved. There are some aspects of the model that are difficult to encode in SBML. For example, the model makes heavy use of vectors to represent regular structures, such as, the chromosome. In SBML, you need a species for each position in the chromosome. This does not scale, since it requires hundreds of thousands of species with unique IDs and duplicated common properties. Although SBML cannot currently represent such structures efficiently, the SBML *arrays* package has been proposed to overcome such a limitation by enabling the construction of complex biological models in a standard manner. The draft specification is currently under review by the community, and this work describes a prototype simulator that uses arrays.

The field of synthetic biology and the *genetic design automation* (GDA) tools that support this field also require efficient modeling. The *arrays* package can also be helpful in this domain. In particular, in synthetic biology, one may be interested in modeling a population of cells that include a genetic circuit design. This abstract uses as an example population of cells with repressilator circuits [1]. Arrays allows the creation of any arbitrary number of cells containing the repressilator circuit.

Although arrays are useful in constructing more concise models, efficiency of analysis is not improved if the arrays are simply "flattened" out for simulation. This abstract describes a simulation method within the GDA tool *iBioSim* [5] that handles array constructs on-the-fly rather than flattening the model. Although this method has some overhead with index calculations, it provides significant memory advantages over simulating flattened models.

2. MODELING

iBioSim provides a user interface to construct SBML models. This includes all SBML core constructs: Compartments, Species, Reactions, Parameters, Rules, Events, Constraints, etc. Figure 1 shows an example of an SBML model con-

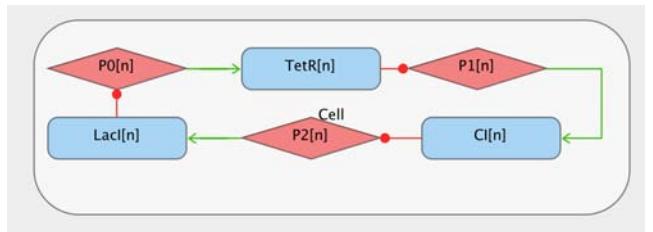


Figure 1: Array of repressilator circuits in *iBioSim*.

structed in *iBioSim*. This model corresponds to the repressilator circuit. In the repressilator, there are three proteins produced from three promoters in which each protein acts as a transcription factor for one promoter creating a loop that forms an oscillator. Namely, the first protein, LacI, inhibits the transcription of the production of the second, TetR, which inhibits the production of the third protein, CI, which inhibits the production of LacI. The tool provides high-level constructs to represent genetic circuits using SBML core elements. The blue rectangles represent chemical species, which in this particular model are the proteins TetR, LacI, and CI. Promoters are represented as red diamonds, which in SBML are simply species. The red arcs represent repression and the green arrows represent genetic production. These processes are represented using reactions in SBML. In addition, the model includes a degradation reaction for each species that is not shown on the figure.

The *arrays* package allows the expression of regular constructs more efficiently. Using this package, SBML objects are extended with the addition of dimensions and indices. An SBML object is an array when it is given dimensions. Each dimension can have a name and an id. Dimensions are required to have a size that points to a constant parameter with a non-negative integer value. In addition, dimensions should have an integer associated with the referred array dimension. Furthermore, objects can have indices to specify an array index to reference an array object. Indices reference array objects by specifying an attribute.

iBioSim has been extended to support such array constructs. Figure 1 shows how this is currently done in *iBioSim*. In order to indicate an object has dimensions, you can specify it by using square brackets enclosing the id to a constant parameter e.g. "[size]". In this particular case, the species TetR, CI, and LacI, and the promoters P0, P1, and P2 are

arrays of size n . The index math is specified within the attributes box of each object. Without arrays, the same population of genetic circuits would have to be instantiated explicitly multiple times. Not only would this be a tedious process, but also it would not scale, since the model would grow quickly for large population sizes.

3. ANALYSIS

There are two ways to simulate models with arrays. The first way is to flatten a model before simulation. The arrays package is just syntactic sugar for SBML models. That is, semantically equivalent models can be constructed without this extension. This means an SBML document using arrays can be flattened into a new SBML document without arrays. This flattening procedure has been implemented in the Java-based library of SBML called JSBML. This routine eases the integration of the arrays package into existing analysis tools. However, this approach has some limitations: it restricts arrays objects to be statically computable (i.e. constant sizes), and it can cause model blow up. The flattened method can be analyzed using a variety of different methods, such as Gillespie's *stochastic simulation algorithm* (SSA) [2].

One of the problems with the flattening approach is that when a model is flattened to an intermediate representation, valuable information is lost. Another way is to simulate the original model as it is. An extension to SSA, where arrays are handled on-the-fly, is implemented within **iBioSim**. In this approach, only one copy of each construct is necessary with the exception of variables. Variables, such as Species, Parameters, and Compartments, among others, need a record of the state of each member of the array. Other constructs need a record of the size of the array. When performing arrayed reactions, events, rules, and others constructs that change the state of the simulation, the simulator iterates through each of the components of the array and performs the necessary updates. Objects that reference other arrayed objects need to calculate the index for each object being referenced. The extended SSA potentially allows for the analysis of arrayed models with dynamically changing sizes. In addition, this approach prevents the model blow up caused by flattening.

4. RESULTS AND DISCUSSION

Comparison between the two approaches is conducted by simulating the arrayed version of the repressilator with different sizes. While the runtime is a bit higher as shown in Fig. 2, the method discussed in this abstract reduces the memory usage substantially. This enables the simulation of larger models. Furthermore, handling arrays on-the-fly ultimately allows one to leverage sparse arrays to further improve the memory-efficiency. In the future, this new simulator could also support dynamic arrays enabling the simulation of populations of cells as they grow and divide.

Acknowledgements

The authors of this work are supported by the National Science Foundation under Grant No. CCF-1218095. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

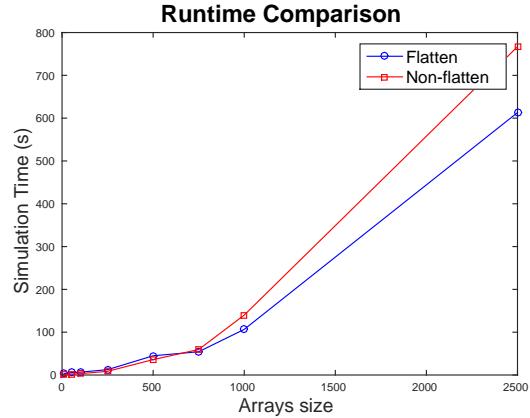


Figure 2: Runtime comparison of both approaches.

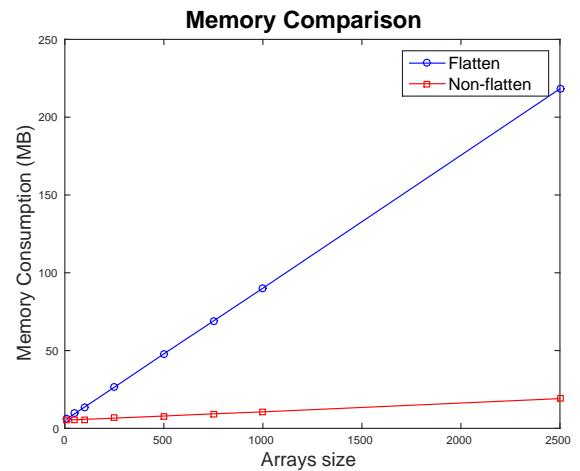


Figure 3: Memory comparison of both approaches.

5. REFERENCES

- [1] M. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.
- [2] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
- [3] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, and et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, Mar. 2003.
- [4] J. R. Karr, J. C. Sanghvi, D. N. Macklin, M. V. Gutschow, J. M. Jacobs, J. Bolival, Benjamin, N. Assad-Garcia, J. I. Glass, and M. W. Covert. A whole-cell computational model predicts phenotype from genotype. *Cell*, (2):389–401, 2015/04/15.
- [5] C. Madsen, C. Myers, T. Patterson, N. Roehner, J. Stevens, and C. Winstead. Design and test of genetic circuits using iBioSim. *Design and Test of Computers, IEEE*, 29(3):32–39, 2012.

Extending the features and improving the performance of gro simulator: new bacterial conjugation and gene expression modules

Extended Abstract

Martín Gutiérrez

Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid
mgtierrez@fi.upm.es

Escuela de Informática y Telecomunicaciones
Universidad Diego Portales
martin.gutierrez@udp.cl

Paula Gregorio-Godoy
Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid
paula.gregorio@upm.es

Guillermo Pérez del Pulgar
Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid
guillermo.pdelpulgar@upm.es

Alfonso Rodríguez-Patón
Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid
arpaton@fi.upm.es

ABSTRACT

gro is a flexible cell programming language and powerful Agent-based Model (AbM) tool developed in Klavins Lab for simulating bacterial colony growth and cell-cell communication. It is used as a prototyping tool in synthetic biology. Here, we present some extensions made to **gro** to: 1) improve its performance (a new shoving algorithm and a new genetic expression module) and 2) add new functionalities (bacterial conjugation and nutrient uptake). The increase in performance is achieved mainly by the shoving algorithm that seeks to minimise calculations for relocating and reorienting simulated bacteria in the colony. The new genetic module also speeds up execution by calculating protein concentrations in each bacterium using a model based on Boolean networks with delays. The overall speedup achieved is about 20-fold. The new **gro** simulator with these modules can grow bacterial colonies of around 10^5 cells in 10 minutes. Bacterial conjugation is being included as a new cell-cell communication method to be simulated with **gro**. The nutrient uptake module treats bacterial growth as an emergent property dependent on underlying nutrient concentration, consumption and biomass conversion.

Keywords

Agent-based Model, Individual-based Model, **gro**, Synthetic Biology, Bacterial Conjugation, Cell growth, Gene expression

1. INTRODUCTION

Synthetic Biology in prokaryotic cells is moving from single-cell programming, where all the cells execute the same genetic program, to distributed synthetic multicellular frameworks with different bacteria processing different programs and communicating with their neighbour cells. Most of the

current multicellular systems are based on quorum sensing cell-cell communication protocols [1]. Other possibilities for intercellular signaling are phage systems [2] or plasmid conjugation [3]. We add conjugation [4] as a new communication protocol for multicellular programming to the features of **gro**. This multicellular approach yields more sophisticated and complex genetic circuits. However, finding the right parameters for optimal functioning and “debugging” the circuits is time consuming and a costly task if done in a wet lab. Computer-based simulators help to fulfil these tasks to a certain extent. Agent-based Model (AbM) is a class of simulators in which each entity of the simulation is treated as an agent, and therefore, mostly operates individually according to its designated behavior. Examples of AbMs for synthetic biology are described in [3, 5, 6, 7]. **gro** [5] is an AbM framework for specifying and simulating multicelled behaviors. It focuses on bacterial colony growth and signalling. It is a powerful tool with a flexible and simple architecture. Improvements such as performance, more communication primitives and a biological-oriented programming paradigm would help to handle more complex and realistic simulations.

2. THE **gro** EXTENSIONS

We now describe the new features that we are including and their impact on **gro**’s functionality.

2.1 CellEngine

CellEngine is a new algorithm for executing bacterial shoving. It replaces Chipmunk [8] in **gro** for that function. This replacement gives rise to an order of magnitude in execution speedup. For example, simulating the growth of a 5000 bacterial cell colony took **gro** about 7 minutes when using Chipmunk, while it only took 15 seconds with CellEngine. This new algorithm will be described in detail in an upcoming work by our group.

2.2 Nutrient uptake module

Bacterial growth in `gro` is driven by a parameter that determines the volume each bacterium gains per minute. However, it does not account for underlying nutrient conditions, such as local nutrient depletion and each of the bacterium's growth phases. We are building a library that implements nutrient uptake in a manner similar to [9]. It is based on a grid that stores the nutrient concentration at each location. Each bacterium consumes a certain amount of nutrients from the location of the grid where it currently is and converts the consumed nutrient to biomass. When no nutrient is left at the specific location, no growth occurs. This module's goal is to increase bacterial growth realism in `gro`.

2.3 Gene expression module

`gro` specifications follow a rule-based paradigm. These rules take the form of guarded commands in which a guard G is tested and if it evaluates to true, a block of instructions B is executed. We propose a complementary approach in which the specification is expressed as a set of gene-expression units grouped into plasmids. The activation and repression of each gene is computed through a boolean network model with delays, an efficient simplification of Differential Equation models. This new gene expression module will allow simple linkage of SBOL 2.0 [10, 11] as an input format for the genetic design for `gro`. The goal of this module is two-fold: to simplify experiment specification and to accelerate execution of the simulations.

2.4 Conjugation

Bacterial conjugation is a Horizontal Gene Transfer mechanism that transfers plasmid DNA from a donor bacterium to a neighboring recipient bacterium. We implemented conjugation in a general manner: depending on the values of two parameters, conjugation frequency, cf , and a neighbor saturation threshold, N_c , conjugation may be set to act as frequency-based, density-based or any intermediate stage [12]. The implementation of conjugation broadens the variety of cell-cell communication primitives available in `gro`. This feature is crucial for simulating multicellular genetic circuits based on conjugation that are being engineered for the EU project PLASWIRES, <http://www.plaswires.eu>.

3. CONCLUSIONS AND FUTURE WORK

We have presented a brief description of the extensions that we are preparing for `gro`: implementation of bacterial conjugation, inclusion of a different programming paradigm for specifying genetic circuits, coupling of bacterial growth with nutrient uptake and the enhancement of `gro`'s performance through a new shoving algorithm. With the new extensions it is possible to specify and simulate a broader range of circuits in `gro` and to perform longer, more complex and realistic simulations. Finally, we conclude that performance bottlenecks for `gro` are located at the physics engine and the program parsing and processing module.

`gro`'s modular architecture enables relatively straightforward addition of more biological tools. We foresee the inclusion of tools such as phages, CRISPR, small RNA regulation, etc. Direct import/export of the models is another interesting direction: Use of SBOL 2.0 [11] enriched with quantitative modelling data could serve as the basis of a standard description language for AbM simulations. We believe that

`gro` can evolve to become even more intuitive and scalable to be used in the realistic simulation of many synthetic biology circuits.

4. ACKNOWLEDGMENTS

This work was partially funded by EU FP7 Project 612146 - PLASWIRES grant, Spanish National Project TIN 2012 - 36992 grant and partially by Becas Chile grant. We also acknowledge BDAC for providing part of the funds to present this work in IWBDA 2015.

5. REFERENCES

- [1] Javier Macía and Ricard Solé. How to make a synthetic multicellular computer. *PLoS ONE*, 9(2), 2014.
- [2] Monica E Ortiz and Drew Endy. Engineered cell-cell communication via DNA messaging. *Journal of Biological Engineering*, 6(1):1–12, 2012.
- [3] Angel Goñi-Moreno, Martyn Amos, and Fernando de la Cruz. Multicellular computing using conjugation for wiring. *PloS ONE*, 8(6):e65986, 2013.
- [4] Irene del Campo, Raúl Ruiz, Ana Cuevas, Carlos Revilla, Luis Vielva, and Fernando de la Cruz. Determination of conjugation rates on solid surfaces. *Plasmid*, 67(2):174–182, 2012.
- [5] Seunghee S Jang, Kevin T Oishi, Robert G Egbert, and Eric Klavins. Specification and simulation of synthetic multicelled behaviors. *ACS synthetic biology*, 1(8):365–374, 2012.
- [6] Timothy J. Rudge, Paul J. Steiner, Andrew Phillips, and Jim Haseloff. Computational Modeling of Synthetic Microbial Biofilms. *ACS Synthetic Biology*, 1(8):345–352, August 2012.
- [7] Laurent A Lardon, Brian V Merkey, Sónia Martins, Andreas Dötsch, Cristian Picioreanu, Jan-Ulrich Kreft, and Barth F Smets. iDynaMiCS: next-generation individual-based modelling of biofilms. *Environmental microbiology*, 13(9):2416–34, September 2011.
- [8] S. Lembecke. *Chipmunk Physics Engine* - <http://chipmunk-physics.net>, 2013 (accessed April 16, 2015).
- [9] Stephen M Krone, Ruinan Lu, Randal Fox, Haruo Suzuki, and Eva M Top. Modelling the spatial dynamics of plasmid transfer and persistence. *Microbiology*, 153(8):2803–2816, 2007.
- [10] Michal Galdziński, Kevin P Clancy, Ernst Oberortner, Matthew Pocock, Jacqueline Y Quinn, Cesar A Rodriguez, Nicholas Roehner, Mandy L Wilson, Laura Adam, J Christopher Anderson, et al. The synthetic biology open language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nature biotechnology*, 32(6):545–550, 2014.
- [11] Nicholas Roehner, Zhen Zhang, Tramy Nguyen, and Chris J Myers. Generating systems biology markup language models from the synthetic biology open language. *ACS Synthetic Biology*, 2015.
- [12] H McCallum, Nigel Barlow, and Jim Hone. How should pathogen transmission be modelled? *Trends in Ecology & Evolution*, 16(6):295–300, 2001.

Phagebook

A Software Environment for Social Synthetic Biology

Kathleen Lewis, Inna Turshudzhyan, Kara Le Fort, Nicholas Musella, Nicholas Roehner, Prashant Vaidyanathan, Douglas Densmore

Department of Electrical and Computer Engineering, Boston University, Boston, MA, USA
{kmlewis, innatur, klefort, nmusella, nroehner, prash, dougd}@bu.edu

1. INTRODUCTION

Synthetic biologists forward engineer living systems to create novel solutions for many of society’s grand challenges in human health, materials, energy, and environmental remediation. This process requires a highly interdisciplinary set of skills and participants. The efficient exchange of information and ideas can be facilitated by connecting researchers to a social networking platform that encourages sharing, data exchange/standardization, and collaboration. Unlike traditional social media experiences (e.g. Twitter, Facebook), a system for synthetic biology will require that the social interaction aspects be tied to a formalized set of design tools that naturally expose real-life design activities to a social interface. Interacting with the social networking tools should enhance research, which in turn should enhance social networking.

Clotho 3.0 is a database management tool for synthetic biology [2]. Phagebook is a Clotho 3.0 app that serves as a social networking interface incorporating lab inventory management and project/personnel networking. Phagebook facilitates collaboration amongst the synthetic biology community and like any other Clotho app, it directly communicates with Clotho apps for specification, design, assembly, and verification of synthetic biological systems.

Phagebook currently consists of three general networking areas: personal, inventory, and research. Its social media aspect allows users to create a profile, connect with other researchers, update publications, and post progress updates for projects and papers.

2. FEATURES

Phagebook uses Clotho to store its data in a MongoDB database. Phagebook utilizes built-in Clotho features to store and manage data. Additionally Phagebook exchanges information with various Clotho applications, such as Phoenix (www.cidarlab.org/phoenix) and Cello (www.cellocad.org). This integration allows users to easily incorporate Phagebook into existing aspects of their research.

2.1 Personal

Phagebook allows users to set up a profile, which lets them be a part of institutions, labs and other social organizations. Using their profiles, Phagebook users can update their status, share publications, receive updates about orders and projects, and add important events to their calendar. Users can “add friends” and collaborators to receive updates from

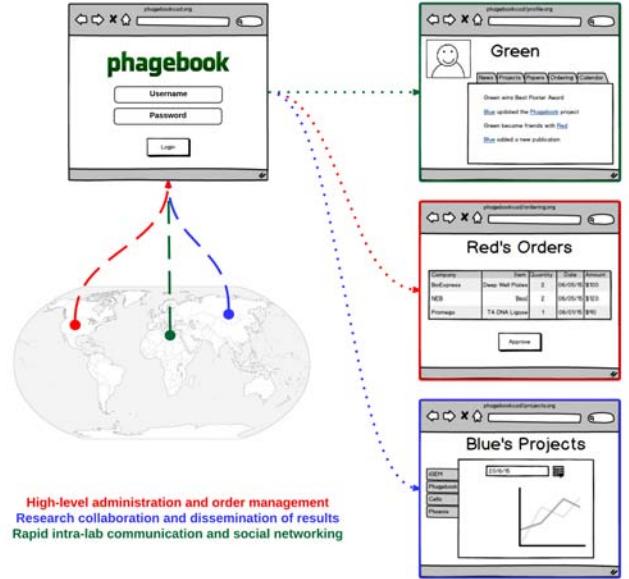


Figure 1: Phagebook allows users to share information via a social networking application. This application is unique in that it leverages real data and existing design tools to inform and enhance the social media experience.

other researchers and keep up with the progress of their colleagues. Phagebook’s integration in Clotho provides a powerful platform allowing users to post statuses with features they have created in Clotho applications, such as DNA sequence data and experiment protocols, which can then be accessed by other users via Clotho.

2.2 Research

Users have the ability to design, create, and share wetlab and computational synthetic biology projects. Projects can be associated with multiple Phagebook members and labs, which can help collaborative efforts. Wetlab projects include online notebooks in which users can record protocols, experiments, genetic parts information, and test results. These notebook entries respect privacy options set by the user and can only be accessed by members of the project. The ability to document and share project details enables users to receive quick updates and enables Lab PIs to keep track of the work and progress being made in a project. Additionally, lab

members have the ability to comment on results, progress updates, experiment protocols and notebook entries.

Phagebook provides an online ecosystem for synthetic biologists. While there are several project documentation tools currently available, such as the iGEM registry [1] and OpenWetWare (www.openwetware.org), Phagebook drives its strength from Clotho's integrated features and metadata capabilities. These unique features allow a more holistic approach to synthetic biology research.

2.3 Lab-Inventory Management

The Phagebook lab-inventory management system grants users easy access to an ordering portal, which lists vendors and product details. Depending on the user's level of authorization, the user can create a new order or edit, approve, and submit an existing order. This simplifies and stream-

lines the ordering process by containing all orders in one accessible location. Additionally, users can monitor inventory and budgets for labs and projects. There are several built-in features that alert a user of budget constraints and recent orders before a new order is placed.

2.4 For more information

See <http://www.cidarlab.org/phagebook>

3. REFERENCES

- [1] K. M. Müller and K. M. Arndt. Standardization in synthetic biology. In *Synthetic Gene Networks*, pages 23–43. Springer, 2012.
- [2] S. Paige, P. Vaidyanathan, M. Bates, J. Anderson, and D. Densmore. Clotho 3.0: An improved common framework for synthetic biology computing. poster presented at Synthetic Biology Boston, June 2014.

Phoenix: An automated design-build-test tool

Evan Appleton, Yash Agarwal, Zachary Chapasko, Ernst Oberortner, Alan Pacheco,
Prashant Vaidyanathan, Nicholas Roehner, and Douglas Densmore

Center of Synthetic Biology, Boston University, Boston, MA, USA

{eapple, yash1214, ztc, pachecoa, prash, dougd}@bu.edu, {nicholasroehner, e.oberortner}@gmail.com

1. INTRODUCTION

The field of synthetic biology has grown considerably in recent years. This growth has improved the core technologies used to engineer genetic systems and also increased the scale of data used to design and build such systems. These increases in scale necessitate the building of tools to store and manage large amounts of data in addition to tools that can reason with this data to improve foundational technologies. As a consequence, there has been significant growth in both the theory and the software tools to accomplish some of these tasks [3], such as standardized data model tools, data storage tools, DNA assembly tools, system design tools, design specification tools, simulation tools and data visualization tools. Although there are currently many tools to solve individual sub-problems, there still remains some important problems that have not been defined and solved in great detail such as design verification and experimental design for data acquisition. Furthermore, there are few tools that successfully integrate tools for each of the sub-problems in a cohesive way such that a user can proceed through iterative design-build-test cycles that do not require detailed knowledge of the tools on the back end.

To help address these problems we have built a tool called *Phoenix*, where users are guided through an iterative, hierarchical design-build-test cycle with sets of building and testing instructions. In this work, we have defined and addressed existing sub-problem gaps and integrated other existing tools and data standards. Information flow in *Phoenix* is presented to a user in an abstracted form such that minimal knowledge of sub-problems is required and an iterative design process can be tracked in a user interface.

2. RESULTS

2.1 Tool overview

At the highest level, a user inputs their design specifications, any existing constraints on those specifications (in terms of structure, function, and performance), and their DNA parts library. Then, *Phoenix* returns iterative sets of instructions for building and testing the genetic constructs. A user can then execute the experiments according to the instructions and return sequence and flow cytometry data, which can then be used to determine the next set of constructs to be built and tested (**Fig. 1**). This iterative process continues until a construct that satisfies the initial design specification input is produced.

The user interface of *Phoenix* displays an interactive map of experiment sets, such that iterative design-build-test phases do not loop endlessly into failed local maximums. Since each experiment is mapped to data from sets of characterized genetic constructs, processed data and simulations can be viewed in the user interface to evaluate processed data

and simulation results. This software framework allows for the bulk of design choices, data analysis and data storage to be handled by computational tools on the back-end, while presenting the user with only the minimal necessary information to evaluate and re-direct designs if necessary.

2.2 Back-End Tool Flow

Phoenix combines seven existing software tools and introduces more than four additional sub-problem definitions and solutions. A majority of these tools are not presented directly to a user, but are used extensively on the back end.

The first series of tools and algorithms goes from an input design specification and parts library to the first set of experimental instructions. This starts with uploading all DNA sequences into a sequence editor tool (*Benchling*¹) and exporting annotated sequences into a multi-part Genbank file. This file, along with files for sets of constraints used to produce abstract genetic regulatory networks (AGRNs) using *miniEugene* [4] are uploaded and saved into *Clotho*² (**Step 0**).

Phoenix is then idle until it receives a set of design specifications from the user (**Step 1**) in the form of a temporal logic equation ('function'), a constraint set to describe candidate structures that can be used as building blocks for complex function ('structure'), and a set of limitations on the observed function ('performance'). These specifications are verified and decomposed using structure- and function-based grammars within *Phoenix* (**Step 2**). These decomposed genetic structures and functions are supplemented with temporary testing parts and the first round of optimized part assignment is performed to get the constructs for the first testing phase (**Step 3**). These target parts and the parts library are then exported into *Raven* [2] to produce an optimized DNA assembly plan (**Step 4**). At this point, the user is returned a file with instructions for DNA assembly, oligos needed for assembly, and a 'key file' that represents the testing needed for subsequent data analysis (**Step 5**) after building is complete.

The user then executes the building and testing instructions and returns into *Clotho* the sequence data via the sequence editor and the annotated key file (**Step 6**). Raw cytometry data is then processed by a data analysis script using the *Bioconductor*³ library in R (**Step 7**). This processed data is then mapped with the key file to the core *Phoenix* mechanistic model, where an algorithm for parameter estimation is used to determine measured rate constants for each construct according to the assumptions of the core model (**Step 8**).

At this point, based on the design hierarchy created by

¹<https://benchling.com>

²<https://clothocad.org>

³<http://www.bioconductor.org>

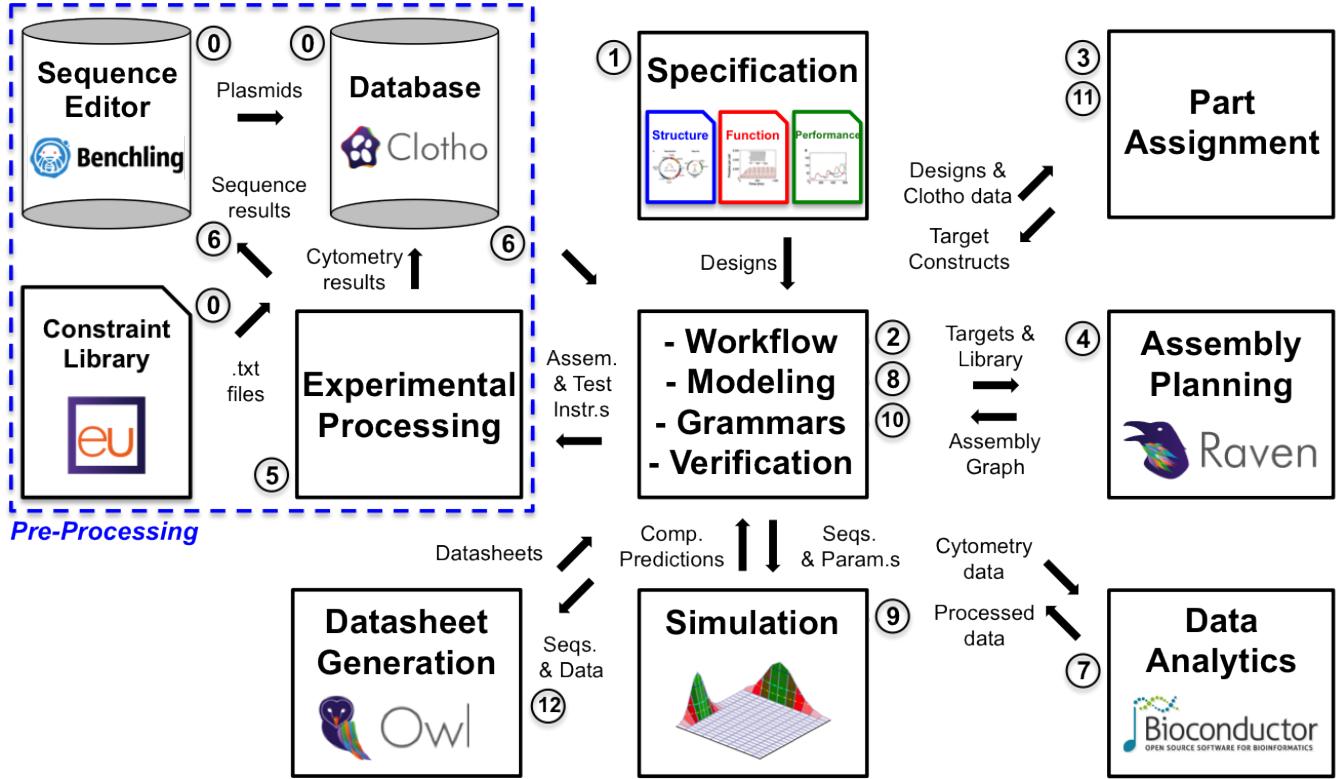


Figure 1: *Phoenix* tool map and flow

the structural and functional grammars upon input, all parameters for the current stage of the design hierarchy are known and can be used to simulate behavior at the next stage. These parameter sets are collected and the parameterized models are converted to an SBML master equation to simulate the behavior of this next stage of the design hierarchy (**Step 9**). Next, these simulations are verified using the decomposed temporal logic equations and those constructs which do not satisfy the function under the bounds of the specified performance at that level of the hierarchy are filtered out (**Step 10**). Lastly, the constructs that have the best verification results are mapped back to parts and another round of part assignment optimization (**Step 11**) is performed and the results of both simulation and observed behavior can be viewed in electronic datasheets using *Owl* [1] (**Step 12**).

Steps 4-12 are repeated iteratively up the design hierarchy determined by the *Phoenix* grammars (**Step 2**), but the user only ever views building end testing instructions (**Step 5**), a graphical map of the design hierarchy (**Step 2**) and results in the user interface (**Step 12**).

3. CONTRIBUTIONS

Phoenix is designed to perform all of the necessary decisions required in a design-build-test cycle for genetic regulatory networks, while only showing a user the minimal information to perform the directed experiments and view simulated and observed results. In addition to the work to identify and link all necessary tools for an automated design-build-test cycle, we created grammars for design decomposition, cloned libraries of DNA building blocks to diversify

construct design space, developed standardized experimental methods for building and testing genetic constructs, developed scripts for parameter estimation based on cytometry data, developed software for functional verification, developed algorithms for part assignment based upon functional outcomes and predictions, and, created a front-end interface that enables tool use without requiring detailed knowledge of each of the tools used on the back end. We intend for this tool to enable users to build highly functional synthetic genetic regulatory networks with minimal background knowledge.

4. ACKNOWLEDGMENTS

The authors would like to thank Traci Haddock, Sonya Iverson, Diego Cuerda, and Katie Lewis for helpful conversations regarding experimental workflows and cloning.

5. REFERENCES

- [1] APPLETON, E., ET AL. Owl: Electronic datasheet generator. *ACS Synthetic Biology* 3, 12 (2014), 966–968.
- [2] APPLETON, E., TAO, J., HADDOCK, T., AND DENSMORE, D. Interactive assembly algorithms for molecular cloning. *Nat Methods* 11, 6 (2014), 657–662.
- [3] GALDZICKI, M., ET AL. The synthetic biology open language (sbol) provides a community standard for communicating designs in synthetic biology. *Nat Biotech* 32, 6 (2014), 545–550.
- [4] OBERORTNER, E., AND DENSMORE, D. Web-based software tool for constraint-based design specification of synthetic biological systems. *ACS Synthetic Biology* (2014).

Pooled, *in situ* Assembly of Complex Genomic Libraries Using Sorter-Assisted Genome Engineering

Robert G. Egbert
UC Berkeley, LBNL

egbert@berkeley.edu

Eric H. Yu
UC Berkeley

ericyu3@gmail.com

Adam P. Arkin
UC Berkeley, LBNL

aparkin@berkeley.edu

ABSTRACT

Engineering diverse genomic libraries of complex biosynthetic pathways and dynamical systems in bacteria, including *E. coli*, is limited by poor recombination efficiency of multi-kilobase linear DNA. We have developed a novel methodology that is compatible with ribosome binding site or coding sequence libraries to integrate multi-gene constructs with high efficiency, allowing extensive sampling of functional variation for engineering biological systems. The method couples a serial “inchworm” chromosomal integration workflow for cycling antibiotic markers with FACS enrichment of pooled genomic libraries that minimizes off-target integrations which contaminate the serial integration process. We have validated the genome engineering approach by integrating a diverse expression library for the 7.5 kb violacein pathway and developed a pooled enrichment screen for antibacterial activity via competition with *B. subtilis*. We have also developed Goldenworm, a software tool that automates primer and integration cassette design, distilling best practices for cassette assembly and high-efficiency recombination. This genome engineering methodology enables integration of diverse, multi-gene genomic libraries suitable for optimizing biosynthetic gene clusters and dynamical systems *in situ* on bacterial chromosomes.

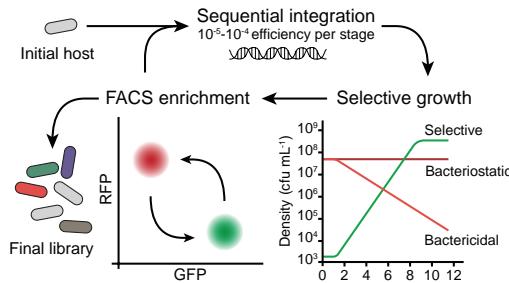


Figure 1. Sorter-assisted pooled library generation workflow includes λ Red-mediated recombination, selective growth and FACS enrichment to sequentially incorporate gene libraries and screen out off-target integrants.

1. INTRODUCTION

Heterologous pathways are introduced into bacteria efficiently via plasmid transformation, generating combinatorial libraries of constructs that are screened or selected for the desired function. However, plasmid-borne constructs have several limitations for production systems or complex environments, including variable copy number and high rates of spontaneous plasmid loss without antibiotics. In addition, modern DNA fragment assembly methods limit library complexity since assembly efficiency is inversely proportional to the number of fragments. Moreover, improper assemblies such as those missing fragments severely limit library

construction, forcing verification at the single colony level. Here we present a method that uses bacterial chromosomes as substrates for the serial assembly of complex genomic libraries that retain library purity through error correction checkpoints inherent to the mechanics of recombination.

2. EXPERIMENTAL RESULTS

We sequentially integrated a five-gene, 7.5 kb violacein production pathway from *Chromobacterium violaceum*. We used the Ribosome Binding Site Calculator [1] to parameterize the predicted expression space of a *vioABEDC* library, with each gene having 8 to 16 RBS variants, giving a potential library size of over 200,000 genotypes. Using five stages of high-throughput electroporation we obtained 10³ - 10⁵ transformants per integration stage, resulting in a final library that samples up to 10% of the designed genotypes.

Our library samples considerable phenotypic diversity in color, titer and fitness when induced, and we sequenced several colonies to verify variation in RBS sequences. Due to the size of the library, we are adapting an emulsion-PCR based assay coupled with high-throughput sequencing [2] to estimate the number of variants.

3. METHODS

We utilized a modified EcNR1 strain [3] with TetR regulated expression of λ Red genes and select exonuclease knockouts to improve recombineering efficiency. Each integration cassette contains a gene library, a selection/enrichment cassette and chromosome homology. Inchworm assembly [4] allows sequential insertion of integration cassettes into a chromosome with strong selection at each stage. Subsequent integrations replace the selection/enrichment cassette from the previous stage, allowing resistance marker recycling and cell sorter enrichment of proper integrants.

Traditional inchworm assembly is not compatible with a pooled genomic library strategy due to the bacteriostatic activity of many common antibiotics and a non-negligible rate of off-target integration. We addressed these limitations by engineering dual fluorescence-selection cassettes using sfGFP and mKate2 with *kanR*, *cmR* and *specR* and using fluorescence-activated cell sorting to screen out off-target integrants. For example, if the stage 1 insert contained RFP and the stage 2 insert contained GFP, cells that are positive for RFP and both GFP and RFP can be removed by cell sorting.

Integration cassettes were constructed using a three-fragment linear Golden Gate assembly followed by gel extraction and nested PCR. The assembled parts consisted of the gene library of interest, a dual fluorescence/antibiotic resistance fusion gene, and a third fragment to introduce 500 bp of homology to the genome. The homology fragment is identical for each integration stage and

serves to boost transformation efficiency. The full-assembly product of Golden Gate assembly was gel extracted and amplified with nested PCR to yield sufficient DNA for high-efficiency transformation (>200 ng).

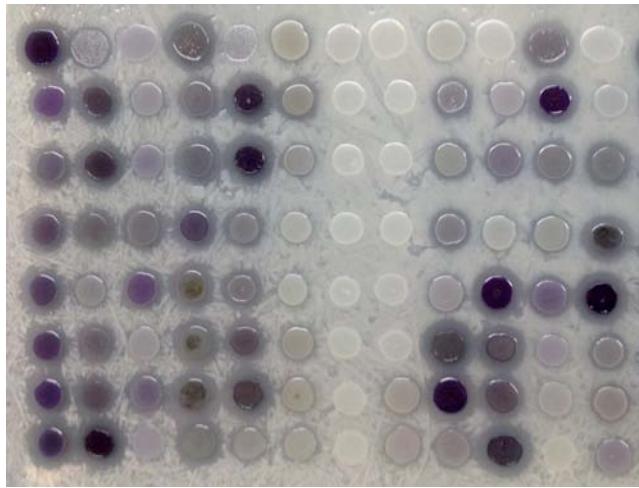


Figure 2. A sampling of full vioABEDC pooled genomic library. The samples represent observed population diversity in growth rate, pigmentation and antimicrobial activities when the pathway is induced.

4. DESIGN AUTOMATION

Our design tool will incorporate the lessons learned from our experimental attempts to improve integration efficiency. For example, protecting the lagging-strand-targeting strand of the DNA fragment with phosphothioate modification has been shown to enhance efficiency [5]; the tool will output integration primers with this modification as appropriate. In addition, the Golden Gate fragment that extends chromosome homology will be designed to integrate at the 3' end of the lagging strand to be compatible with the asymmetric nature of λ Red recombination [5].

The design tool will function similarly to JBEI's j5 [6], accepting a list of (possibly degenerate) parts and generating the required primers, along with annotated GenBank files describing the expected products. Input sequences are given as annotations in GenBank files; sequence context is required to include a leader region in the initial PCR reaction to ensure compatibility with nested PCR, which increases yield and specificity for the integration cassette. The algorithm includes regions of homology to the genome in the integration primers to allow efficient and specific recombination on the chromosome.

The tool will take into account best practices for Golden Gate assembly and warn about potential problems such as internal Type IIs restriction sites or off-target homology. Sticky ends will be designed such that non-adjacent parts have no more than 2 bp in common, and fragment sizes will be chosen to allow easy gel extraction of the product.

The web interface for the tool will be accessible at <http://goldenworm.genomics.lbl.edu> where users can upload sequences and download the generated primers and recommended assembly and integration protocols.

5. CONCLUSION

We successfully developed an efficient and generalizable method for genomic integration of multi-gene libraries in *E. coli*. This method is capable of generating large combinatorial libraries for metabolic pathways, genetic circuits, or any other genetic constructs with design uncertainty that can be tested with genomic libraries. To date we have demonstrated up to five cycles of sequential integration. Future work will investigate experiment-based optimization of fragment size to accommodate designed libraries with minimal losses in efficiency.

The goal of the design tool is to make it easy for researchers to apply our assembly and recombination strategy to their own genomic library design and to allow integration with robotic automation. By providing it to the public, we hope it will accelerate engineering functionally complex systems in microbes.

6. REFERENCES

- [1] Salis, H.M. et al. 2009. Automated design of synthetic ribosome binding sites to control protein expression. *Nature biotechnology*. 27, 10 (Oct. 2009), 946–950.
- [2] Zeitoun, R.I. et al. 2015. Multiplexed tracking of combinatorial genomic mutations in engineered cell populations. *Nature biotechnology*. 33, 6 (Jun. 2015), 631–637.
- [3] Wang, H.H. et al. 2009. Programming cells by multiplex genome engineering and accelerated evolution. *Nature*. 460, 7257 (Aug. 2009), 894–898.
- [4] Santos, C.N.S. and Yoshikuni, Y. 2014. Engineering complex biological systems in bacteria through recombinase-assisted genome engineering. *Nature protocols*. 9, 6 (May 2014), 1320–1336.
- [5] Maresca, M. et al. "Single-stranded heteroduplex intermediates in λ Red homologous recombination." *BMC molecular biology* 11.1 (2010): 54.
- [6] Hillson, N.J. et al. 2012. j5 DNA assembly design automation software. *ACS synthetic biology*. 1, 1 (Jan. 2012), 14–21.

Scylax™ - Automated design of cell factories incorporating synthetic enzymes

Michal Galdzicki¹, Kyle Medley², Rudesh D Toofanny¹, Stanley Gu², Yih-En Andrew Ban¹, Herbert M Sauro², Alexandre Zanghellini¹

¹Arzeda Corp 2715 W Fort St, Seattle, WA

²University of Washington, Department of Bioengineering, Seattle, WA 98195

ABSTRACT

We developed Scylax™ as a set of computational tools to design metabolic pathways from a starting metabolite to molecules of value to the chemical industry. To do so requires novel metabolic pathways not existing in nature and that are optimized for chemical production. The resulting pathways are rationally designed and novel, incorporating synthetic enzymes. At Arzeda, our ability to rationally design novel enzymes for potentially any reaction opens new avenues for cell factory design. Scylax™ will enable us to produce fine chemicals that can be difficult to synthesize and to ferment biomass at a scale required for our industrial needs.

1. INTRODUCTION

Arzeda genetically engineers yeast to make molecules of value to the chemical industry. We do so by engineering novel metabolic pathways in yeast to convert feedstock into the product. Our distinguishing characteristic is that we design enzymes that catalyze reactions not found in nature. By incorporating computationally designed enzymes we can propose pathways from a broader search space than competitors, who rely on expression of heterologous enzymes. Furthermore, we can optimize for a high yield of product and offer unique solutions not available otherwise. Our potential solutions are tested experimentally to create novel cell factories, that are further developed in partnership with (petro)-chemical companies. However, the potential solution space for any given product of interest requires a computational methodology to identify putative pathways. In collaboration with KM, SG and HMS at the University of Washington we developed Scylax™ a set of databases, a computational engine and analysis interface (Redox UI) to find all pathways employing natural and designed enzymes for a given product.

Initial steps in this direction have been taken previously. For instance, Mavrovouniotis[1] and more recently Hatzimanikatis and colleagues[2] have developed computational approaches to systematically explore the diversity of complex metabolic routes from a starting point to a product. However, the methods are based solely on the type of chemistry that existing enzymes are known to catalyze (for instance, carbonyl reduction) and remain purely theoretical since the likelihood of (re)designing an enzyme having the desired substrate specificity is not considered.

2. SCYLAX™ automated pathway prediction

2.1 Natural Enzyme Reaction Database

First, we leveraged current known data on enzymes and the chemical reactions they catalyze. We developed our in-house Natural Enzyme Reaction Database (NerdB) by aggregating and curating data from public data resources known enzymes, the EC

classification, the reactions they catalyze, and chemical and structural information for their substrates, and products.

2.2 Generalized Enzyme Reaction database

Next, we created the Generalized Enzyme Reaction database (GerdB) of reaction operators that generalize reactions in the Enzyme Classification (EC) nomenclature to the sub subclass level of EC numbers (e.g. 1.1.1). A reaction operator for EC class 1.1.1, for example, can be applied to any compounds containing a primary alcohol, which converts the functional group on the compound to an aldehyde. The operation tells us the enzyme class that could perform the reaction and will direct the potential design of novel activity for non-natural compounds by using Arzeda's computational enzyme design technology. Each operator is manually curated and validated by Arzeda's expert biochemists enabling prediction of the chemical reaction, as well as the feasibility of design for the putative enzymes.

2.3 Pathway enumeration engine

2.3.1 Algorithm

The goal of the Scylax™ algorithm is to exhaustively find metabolic pathways. The algorithm considers all natural and predicted enzyme reactions from the NerdB and GerdB as putative steps for a pathway. It proceeds by enumerating all pathways from a start compound to the product, subject to filtering methods: pathway length cutoff, and mass yield.

2.3.2 Carbon yield

We developed a method to eliminate paths that do not represent the direct chemical conversion of the starting compound to the product compound. In the majority of cases for metabolic engineering applications this is where the carbon atoms from the starting compound does not end up in the final product. To enable this filtering in the pathway enumeration algorithm we pre-computed the number of shared carbon atoms between every pair of reactants and products. Our algorithm then uses this relationship to traverse only the relationships where the number of carbons shared is greater than a parameter value provided for each run. Adding a minimum cutoff of at least one carbon to filter the number of pathways returned has a significant effect. As an example of the effect of the carbon yield cutoff on the number of viable pathways returned from glyceraldehyde 3-phosphate to pyruvate and 3-phospho-D-glycerol phosphate to pyruvate is reduced by around 75-80% (where 100% represents the number of compounds returned where the carbon yield is 0).

2.3.3 Top down and bottom up enumeration

The combinatorics of running the complete enumeration from D-glucose 6-phosphate to pyruvate to recover even the shortest natural pathway of 8 steps would take us many days to complete. Pyruvate is known to play a key role in many major metabolic pathways and hence enumerating metabolic pathways that involve

pyruvate will return many solution since pyruvate is a highly connected compound. In a benchmark test for glycolysis we enumerated the last 5 steps of glycolysis – namely glyceraldehyde 3-phosphate to pyruvate. To get an idea of the level of connectivity pyruvate has, we looked at the number of resulting pathways that were returned (with the minimum carbon yield set to zero), this enumeration returned in 73,031 pathways. Conversely, the first 4 steps of glycolysis going from D-glucose 6-phosphate to 3-phospho-D-glyceroyl phosphate only report 145 pathways. Hence, we applied a simultaneous bottom-up and top-down search with a limited number of enzymatic steps to more efficiently identify the pathways from D-glucose 6-phosphate to pyruvate Figure 1. This approach requires the enumeration of all possible paths from a *given* compound for N steps. We repeat this enumeration for both the starting and ending compounds as the *given*. Finally, we create complete pathways by finding the overlapping compounds from both searches and enumerating the paths that result from their combination. Using this approach, we were able to recover the shortest natural pathway of 8 glycolysis steps along with the other expected natural pathways.

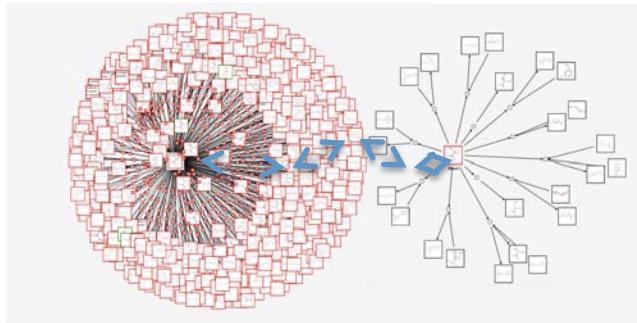


Figure 1. Mockup (actual results too numerous to display) demonstrating pathways finding using top down bottom up approach used for highly connected compounds.

2.3.4 Thermodynamic feasibility of pathways

Thermodynamic profiles based on our implementation of Jankowski's group contribution method[3] of estimating the free energy of formation allow us to discriminate the putative paths. Cumulative ΔG of reaction plotted per step in the path enables us to exclude paths with unfavorable (dark blue line) predicted thermodynamic properties show in Figure 2.

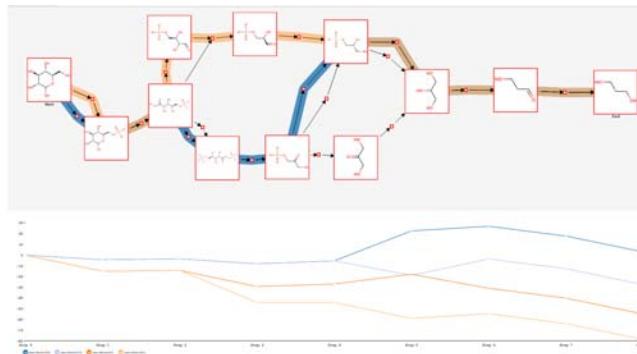


Figure 2. Putative 1,3 propanediol pathways predicted by Scylax™

2.4 Integration and benchmark

We ran a set of benchmark pathways to test the path enumeration by demonstrating the ability to return expected paths present in literature and one in development at Arzeda under an earlier

project. For our benchmark pathways we chose **natural pathways**: ethanol, 2,3-butanediol, and **non-natural pathways**: 1,4-butanediol[4], 1,3-propanediol[5], levulinic acid (Arzeda proprietary pathway). We were able to recapitulate the correct sequence of known enzymes that complete each of the pathways, whilst also enumerating many other potential pathways using existing and novel enzymes.

2.5 Example: fermentation to 1,4-butanediol

We ran the path enumeration to recover the pathways from α -ketoglutarate and succinate to 1,4-butanediol. These are Genomatica's pathways as described in Yim *et al.* 2011[4]. We used this benchmark set of pathways to drive the development of Generalized Enzyme Reaction (GER) implementation. We defined a reaction operator for each step of the pathway as a GER and ran a benchmark test to recover the correct pathway using the full NER and only the GERs that are used in the pathways. We were able to recover the expected pathways from this reduced subset of GERs. As a more comprehensive test we ran the enumeration using the full NerdB and the full GerdB. Initially we found that due to the complexity of applying each reaction operator to the pool of new compounds generated, the enumeration would not complete in a timely manner. We adopted the simultaneous top-down, bottom-up search here also, we were able to recover both reference pathways for 1,4-butanediol pathway, starting from either succinate or α -ketoglutarate.

3. PATHWAY VISUALIZATION

An interactive user interface was needed to aid in the exploration of the numerous pathways generated by Scylax™. Redox UI is a web browser application designed to visualize and iteratively filter the predicted pathways. Redox allows us to distill meaningful results from the raw output of the enumeration algorithm. A screenshot of Redox's rendering of pathways can be seen in Figure 1 and Figure 2.

4. CONCLUSIONS

Our development and benchmarking of Scylax™ establishes a proof-of-concept for automated pathway design that generates libraries of feasible pathways from the information contained in natural and designed enzyme databases. Already, we have begun using the Scylax™ tools at Arzeda to generate leads for targets on projects internally and with partners. Going forward Scylax™ will also help us to refactor pathways with new enzymatic steps or to remove unnecessary steps.

5. REFERENCES

1. Mavrovouniotis, M. L., Stephanopoulos, G. & Stephanopoulos, G. Computer-aided synthesis of biochemical pathways. *Biotechnology and bioengineering* **36**, 1119–1132 (1990).
2. Hatzimanikatis, V. *et al.* Exploring the diversity of complex metabolic networks. *Bioinformatics* **21**, 1603–1609 (2004).
3. Jankowski, M. D., Henry, C. S., Broadbelt, L. J. & Hatzimanikatis, V. Group Contribution Method for Thermodynamic Analysis of Complex Metabolic Networks. *Biophys. J.* **95**, 1487–1499 (2008).
4. Yim, H. *et al.* Metabolic engineering of Escherichia coli for direct production of 1,4-butanediol. *Nat. Chem. Biol.* **7**, 445–452 (2011).
5. Nakamura, C. E. & Whited, G. Metabolic engineering for the microbial production of 1,3-propanediol. *Curr. Opin. Biotechnol.* **14**, 454–459 (2003).

Towards Semi-Automated Experimental Design Using Model Inference in Synthetic Biology

Tileli Amimeur
University of Washington
Seattle, WA
tamimeur@u.washington.edu

Eric Klavins
University of Washington
Seattle, WA
klavins@u.washington.edu

Keywords

Bayesian Computation, Wet Lab Automation, Design Automation, Parameter Inference, Combinatorial Design, Iterative Design

1. INTRODUCTION

The current synthetic biology design-build-test cycle is slow and tedious. Unlike most engineering fields, parts and specifications in synthetic biology are not clearly defined, making reproducibility and optimization in the area a real challenge. Aquarium is a system developed in the Klavins lab for high-throughput, reproducible, semi-automated protocol execution in the wetlab. Our objective is to take advantage of Aquarium's potential for high-throughput experimental characterization, to explore combinatorially-constructed genetic regulatory networks. We are working to create a framework for systematically learning the map from a genetic circuit design space to its behavior space using experimental data. Having such a map can lead to more accurate predictions about the behavior of new synthetic designs, and can therefore provide researchers with a minimal set of experiments for reaching a specific design goal. We introduce a Design Recommendation Tool (DRT) to assist designers with gene network design for a pulse generation circuit using a library of promoters and repressors.

2. DESIGN RECOMMENDATION TOOL

Our envisioned framework of a closed-loop and mostly automated design, build, test cycle in synthetic biology consists of three main portions. The build and test portion of the cycle is implemented by Aquarium. The design portion, the DRT, takes as input a design goal and outputs a set of designs to build. The compile portion, currently being developed by N.bolten and L.Adams also of the Klavins lab, compiles those designs into an actual build strategy which is then sent to Aquarium. Aquarium then sends back any characterization data of those designs back to the DRT.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IWBDA '15 Seattle, Washington USA

2.1 Specifications

The user must enter as inputs to the DRT the design goal specification along with a design grammar specification[1]. The design goal specification is defined as the desired behavior of the system in question at its final output. In addition, the user must provide a definition for the parts of the system and how they fit together structurally. In other words, the user must input the grammar of a design.

The DRT takes as additional input the experimental characterization data of designs returned by Aquarium. The user must specify the type of data it expects in order to meet the design goal requirements. The DRT outputs a set of designs to implement as terminal implementations of the insert grammar G.

2.2 Approach

We are particularly interested in how the parts in a system affect the up-regulation or down-regulation of a gene. Our method is a parameter and model inference approach to learning circuit behavior; as such, the learned model is currently defined as an array of all possible parameters in the system, i.e the parameters to the following master equation:

$$g_a = \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} (D_{a+1,j+1} \cdot 0^{0^{N_{i+1}}} \cdot \frac{L_{j,i}}{1 + k_{i,j} g_i}) - g_a \quad (1)$$

which, when evaluated for $a = 0, 1, 2$, generates a set of ODEs for each design $D_{a,j}$. In the above equation, N is an array representing the number of promoters promoting a specific gene, i.e the sum of each row in D . $L_{j,i}$ represents the expression rate of a promoter P_j promoting gene g_i . The parameter $k_{i,j}$ represents strength of repression by a gene g_i in the design.

The DRT combinatorially assemble the entire design space and generates all the respective ODEs/parameter sets for each design. It begins by doing Bayesian model inference using the *ABC-sysbio* package across all three-cassette designs under consideration [2]. The Joint PDF should show that with no priors/intial knowledge of any parameters any of the potential final designs could have generated the desired behavior data set (Figure 2a). The DRT then chooses the simplest set of designs with the most to learn from (minimizing uncertainty in parameters). It submits these designs to Aquarium to build and test. The DRT uses the returned characterization data from Aquarium and performs Bayesian parameter inference (again using *ABC-sysbio*) on each design. The global parameter priors are then updated and model inference is redone with new parameters. It continues iterating until there is a clear preference for one (or

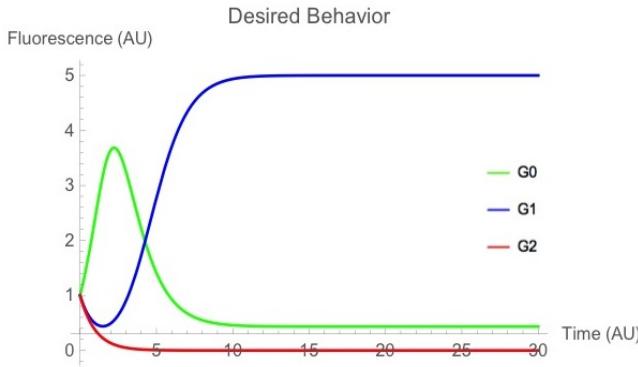


Figure 1: Desired pulse-generating behavior of the GFP reporter (G_0), for a genetic circuit consisting of three genes (G_0, G_1, G_2).

a few) final three-cassette designs for generating our ideal behavior dataset and then build/test those final designs.

3. PRELIMINARY RESULTS

To validate the DRT approach, we developed a test environment to confirm that our tool can learn and uncover the optimal pulse-generating genetic circuit in simulation, having no prior information about how the genetic parts behave, before applying it to true experimental data.

3.1 Test Environment

Our goal is to determine how to optimally construct a synthetic pulse-generating circuit matching the exact behavior specification in Figure 1, using a library of repressors and promoters (the behaviors of which are initially unknown). The rules for composing a design are written in the form of a design grammar to generate the set of all designs that contain only one promoter-reporter cassette followed by any number of promoter-gene cassettes, as can be filled in by the given parts library.

For our preliminary testing we limit the large combinatorial design space to only a few design circuits. In particular, we only have two potential candidates for final designs, one of which (model 1) is the model we used to generate the behavior in Figure 1.

3.2 Uninformed Design Selection

Because the parameter set is so large at the level of three-cassette designs, using our master equation, model selection with uninformed priors on the parameters leads to mixed results and no strong indication of which model could produce our desired behavior. Figure 2a below shows the joint probability distribution resulting from an ABC-sysbio run of our candidate models with no informed priors.

3.3 Informed Design Selection

The DRT's approach to model selection is sequential, and for each iteration attempts to reduce uncertainty in the parameters of the candidate models using the simplest circuits it can build/test. Figure 2b shows clear model selection of model 1 (the model which intially generated the data) after uncovering more definitive parameter priors in previous

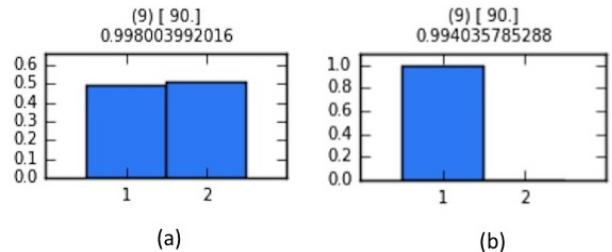


Figure 2: This shows the joint model distributions of the two candidate models to have generated our desired data. Model 1 (the system that actually generated the data) is appropriately chosen only after having had informed priors. Without informed priors, the system cannot determine which model generated the data. (a) Model selection with uninformed parameter priors. (b) Model selection with informed parameter priors, given by iterative DRT parameter inference

rounds of inference.

4. FUTURE WORK

Our current preliminary results is only a first pass at validation of the system. There remains a substantial amount of work to validate the iterative DRT framework. Importantly, a better distance function will need to be used during the inference cycles. We will also need to accomodate much more of the existing design space to really test out the algorithm. Additionally, we are currently working to reproduce the pulse-generating circuit experimentally using the CRISPR system, and are in the process of interfacing the DRT to Aquarium to apply the same model inference framework in a real lab scenario [3]. We hope to use the DRT to aide physical construction of new genetic circuits with a variety of desired dynamic and steady state behaviors.

5. ACKNOWLEDGMENTS

We'd like to acknowledge Chris Barnes, formally of the Theoretical Systems Biology Lab at the Imperial College in London, for our use of the *abc-sysbio* module he developed. Our algorithm makes significant use of this tool which he developed for parameter and model inference.

5.1 References

- [1] Y. Cai, M. Lux, L. Adam, J. Peccoud. Modelling Structure-Function Relationships in Synthetic DNA Sequences using Attribute Grammars. *PLoS Computational Biology*. Vol 5. Issue 10. October 2009.
- [2] J. Liepe, P. Kirk, S. Filippi, T. Toni, C. Barnes, M. Stumpf. A framework for parameter estimation and model selection from experimental data in systems biology using approximate Bayesian computation. *Nature Protocols*. Vol 9. Issue 2. January 2014.
- [3] F. Farzadfar, S. Perli, T. Lu. Tunable and Multifunction Eukaryotic Transcription Factors Based on CRISPR/Cas. *ACS Synthetic Biology*. Vol 2. Issue 10. October 2013.

Towards a sequence-level DNA design specification language

Nick Bolten

Department of Electrical Engineering
University of Washington
Seattle WA
nbolten@gmail.com

Eric Klavins, PhD

Department of Electrical Engineering
University of Washington
Seattle WA
klavins@uw.edu

Categories and Subject Descriptors

D.2.2 [SOFTWARE ENGINEERING]: Design Tools and Techniques

1. INTRODUCTION

CAD (computer-aided design) software enables engineers to rapidly prototype and codify designs within the constraints of their field, including synthetic biology. There are two distinct classes of CAD software in synthetic biology: visual design tools and programmatic libraries and languages [4].

Visual design tools for DNA constructs were already established prior to the development of the field of synthetic biology and have largely consisted of advanced text editors that constrain the alphabet (e.g. A, T, G, and C) used, allow user-defined tagging of functional regions, and display those features on a linear or circular map (VectorNTI [11], Geneious [7], TinkerCell [3], Benchling.com, j5 [8], and Geneious). Visual CAD tools offer a gradual learning curve and immediate visual feedback for design choices and as a result are ideal for exploratory design. The other class of synthetic biological CAD software is domain-specific languages, some visual and some programmatic. These tools aim to provide further levels of abstraction, allowing user-defined semantics, grammars, and constraints to be defined over a set of parts (Eugene [2], GenoCAD [6]). In addition, there are scientific software libraries that are often used for synthetic biological CAD tools such as the many core biological libraries (Biopython [5], BioPerl [12], BioJava [10]). However, because they were originally intended for sequence analysis, common design tasks (for example, primer design and restriction digest) are often unavailable or poorly maintained.

Programmatic tools offer a steeper learning curve than visual CAD tools, but are more expressive in the complexity of what designs can be specified and enable higher levels of abstraction. In addition, source code for biological designs provides a reviewable specification of a design strategy, serving as both documentation and reproducible, extensible software. Despite the power and usefulness of programmatic tools, there is currently no standard library or language for both describing and operating on DNA at the level of exact sequences; existing high-level frameworks such as Eugene, GenoCAD, and Proto [1] leave specification of sequences up to the user. As a result, the core data types and operations on DNA (and RNA) tend to be reimplemented for each project, sitting "under the hood" and largely unavailable to the designer. Despite the lack of a standard programmatic CAD tool for doing so, design at the sequence level is ubiquitous in synthetic biology.

Here, we propose to address this lack of a DNA language by introducing pymbt, a proof of concept Python library that enables the concise expression of sequence-aware data types and the operations on them. By relying on the high-level scripting

language of Python, data types for DNA, RNA, and Peptides have had their common logic and operation functions such as concatenation and subsetting redefined, providing a language-like abstraction layer on top of a popular scientific programming language. pymbt provides a set of data types and functions representing the core processes of synthetic biology that can be (and have been) extended to express custom synthetic biological design algorithms. In addition, pymbt comes with modules representing common operations on nucleic acids that produce new sequences (the reaction module) as well as de-novo sequence generators (the design module)

2. BODY

2.1 Language domain, data types, and core functions

The core data type of pymbt is a sequence (The BaseSequence class) upon which DNA, RNA, and Peptide data types are modeled. Biological sequences are represented as strings (or in the case of double-stranded DNA, sets of strings) for which a limited alphabet is defined: for DNA, {ATGCN}, for RNA, {AUGCN}, and the 20 proteinogenic amino acids for Peptides. Initialization of the data types is simple, requiring only a valid string:

```
my_dna = pymbt.DNA("ATGC")
my_rna = pymbt.RNA("AUGC")
my_peptide = pymbt.Peptide ("MSQQG")
```

Through slight modifications of the base library, DNA and RNA could be made aware of the less-common ambiguous alphabets that include all combinations of A, T, G, and C of size less than three (for example, R, representing any purine (G or C)). All sequences are aware of sets of features-annotations describing subsets (usually functional domains) of the sequence. On top of these basic data structures, all sequences have had their common operators overridden with custom, sequence-aware behaviors. For example only sequences of the same type can be concatenated with the + operator and the subset operation (slicing, done using square brackets in Python []) is aware of both strands of a double-stranded DNA sequence.

```
my_dna + my_dna
[1] ATGCATGC
my_dna[0:2]
[2] AT
```

Concatenating linear and circular DNA has been abolished, as it violates the topology of the strands. Multiplying sequences by an integer results in repetition of that integer and is implemented using the smallest number of concatenations. In addition to overridden operator behavior, the sequences data types include core methods (functions) representing common design operations that produce a new, modified copy of the sequence. For example, the DNA data type includes methods that modify its topology (circular or linear), reverse complement the sequence, calculate the GC content or melting temperature, insert sequences,

searching within a sequence, and compare for equivalence to rotated sequences, among others. In addition to these core sequence data types, specialized cloning types have also been defined to include primers, fragments, and restriction enzymes. From this base, a large variety of designs can be specified with concise syntax, including the built-in reaction and design modules.

2.2 The reaction module

The reaction module includes functions that operate on sequences to produce new sequences, chiefly the processes of the central dogma (transcription, translation) and common cloning reactions (PCR, restriction digest, Gibson assembly, and oligonucleotide assembly). All of the functions are available at the top level of the module, so the syntax for producing a peptide sequence from a (valid) open reading frame is:

```
x.dna = pymbt.DNA("ATGGGAGTGGCGATAG")
x.rna = pymbt.reaction.transcribe(x.dna)
x.peptide = pymbt.reaction.translate(x.rna)

x.peptide
[1] MGVR
```

2.3 The design module

The design module is primarily concerned with generating de novo sequences commonly required for computational sequence design, including uniformly random sequence generation of any length (design.random_dna), codon-biased DNA sequence generation from a peptide (design.random_codons), and melting-temperature and dimer-avoiding based primer design, including automatic Gibson assembly primer generation.

2.4 Other modules

pymbt includes other modules used by the main reaction and design modules or extending pymbt's functionality. They are the analysis module, constants module, database module, and seqio module. They use NuPACK [13], Vienna RNA [9], Rebase, Intermine, Entrez. Finally, the seqio module enables pymbt to read and write sequences in common formats.

2.5 A design example

The following function written with pymbt facilitates the fusion of an ORF consisting of a native *Saccharomyces cerevisiae* (yeast) gene with its stop codon removed to a protein of interest. For this task, the sequences surrounding the stop codon of user-defined genes must be extracted from a set of chromosomes (a list of sequences obtained from the Entrez module):

```
def cterminus_extract(locus_name, chromosomes):
    for chromosome in chromosomes:
        locus_feature = None
        for feature in chromosome.features:
            if feature.name == locus_name:
                locus_feature = feature
        if locus_feature is not None:
            upstream = chromosome[locus_feature.stop - 1000:locus_feature.stop]
            # Remove stop codon from upstream ([:-3])
            while upstream[-3:] in [pbt.DNA("TAG"), pbt.DNA("TGA"), pbt.DNA("TAA")]:
                upstream = upstream[:-3]
            downstream = chromosome[locus_feature.stop:locus_feature.stop + 1000]
    return upstream, downstream
```

This function iterates over each chromosome, looking for a feature with the specified name. Once found, the stop codon is removed and the regions 1000 bp before and after the stop codon are isolated. In downstream code, these sequences were used to generate a small library of integrating plasmids that efficiently

tagged native yeast genes with proteins of interest.

3. CONCLUSIONS

Because the data types begin at basic unit of synthetic biology, biological sequences, pymbt represents a fully bottom-up approach to synthetic biological design. In fact, the re-action and design modules are fully-implemented using only the core data types, constants module, and analysis module, and demonstrate the extensibility of the system. Demonstrating extensibility and expressiveness, pymbt has already been used to generate design algorithms for yeast integration plasmids, yeast knockouts, degenerate peptide linkers, and complete end-to-end plasmid mockup, primer design, construction approach validation, and sequencing result analysis workflows. Because pymbt has been written in Python, it can be (and in some cases, already has been) extended to work with a large ecosystem of existing scientific libraries, enabling diverse sources of data and sequences or even automated submission of designs to genetic part synthesis companies. Finally, by operating at the level of sequence design, pymbt provides a way for researchers to rapidly develop shareable design strategies using operations analogous to those of visual CAD tools.

4. REFERENCES

- [1] J. Beal, T. Lu, and R. Weiss. Automatic Compilation from High-Level Biologically-Oriented Programming Language to Genetic Regulatory Networks. PLoS ONE, 6(8):e22490, 2011.
- [2] L. Bilichenko, A. Liu, S. Cheung, E. Weeding, B. Xia, M. Leguia, J. C. Anderson, and D. Densmore. Eugene: A Domain Specific Language for Specifying and Constraining Synthetic Biological Parts, Devices, and Systems. PLoS ONE, 6(4):e18882, 2011.
- [3] D. Chandran. Tinkercell: computer-aided design for synthetic biology. University of Washington, 2011.
- [4] G. M. Church, M. B. Elowitz, C. D. Smolke, C. A. Voigt, and R. Weiss. Realizing the potential of synthetic biology. Nat Rev Mol Cell Biol, 15(4):289–294, Apr. 2014.
- [5] P. J. A. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, and Others. Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics, 25(11):1422–1423, 2009.
- [6] M. J. Czar, Y. Cai, and J. Peccoud. Writing DNA with GenoCAD. Nucleic Acids Research, 37(suppl 2):W40–W47, 2009.
- [7] A. J. Drummond, B. Ashton, S. Buxton, M. Cheung, A. Cooper, C. Duran, M. Field, J. Heled, M. Kearse, S. Markowitz, and Others. Geneious v5. 4, 2011.
- [8] N. J. Hillson, R. D. Rosengarten, and J. D. Keasling. j5 DNA Assembly Design Automation Software. ACS Synthetic Biology, 1(1):14–21, 2012.
- [9] I. L. Hofacker. Vienna RNA secondary structure server. Nucleic acids research, 31(13):3429–3431, 2003.
- [10] R. C. G. Holland, T. A. Down, M. Pocock, A. Prlic, D. Huen, K. James, S. Foisy, A. Dra'ger, A. Yates, M. Heuer, and Others. BioJava: an open-source framework for bioinformatics. Bioinformatics, 24(18):2096–2097, 2008.
- [11] G. Lu and E. N. Moriyama. Vector NTI, a balanced all-in-one sequence analysis suite. Briefings in bioinformatics, 5(4):378–388, 2004.
- [12] J. E. Stajich, D. Block, K. Boulez, S. E. Brenner, S. A. Chervitz, C. Dagdigian, G. Fuellen, J. G. R. Gilbert, I. Korf, H. Lapp, and Others. The Bioperl toolkit: Perl modules for the life sciences. Genome research, 12(10):1611–1618, 2002.
- [13] J. N. Zadeh, C. D. Steenberg, J. S. Bois, B. R. Wolfe, M. B. Pierce, A. R. Khan, R. M. Dirks, and N. A. Pierce. NUPACK: analysis and design of nucleic acid systems. Journal of computational chemistry, 32(1):170–173, 2011.