Navigating the Maze: The Impact of Configurability in Bioinformatics Software

Mikaela Cashman
Myra B. Cohen
Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE, USA
mikaela.cashman@huskers.unl.edu
myra@cse.unl.edu

Priya Ranjan*
Robert W. Cottingham
Biosciences Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA
pranjan@utk.edu
cottinghamrw@ornl.gov

ABSTRACT

The bioinformatics software domain contains thousands of applications for automating tasks such as the pairwise alignment of DNA sequences, building and reasoning about metabolic models or simulating growth of an organism. Its end users range from sophisticated developers to those with little computational experience. In response to their needs, developers provide many options to customize the way their algorithms are tuned. Yet there is little or no automated help for the user in determining the consequences or impact of the options they choose. In this paper we describe our experience working with configurable bioinformatics tools. We find limited documentation and help for combining and selecting options along with variation in both functionality and performance. We also find previously undetected faults. We summarize our findings with a set of lessons learned, and present a roadmap for creating automated techniques to interact with bioinformatics software. We believe these will generalize to other types of scientific software.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; • **Applied computing** → *Computational biology*;

KEYWORDS

configurability, bioinformatics, software testing

ACM Reference Format:
Mikaela Cashman, Myra B. Cohen, Priya Ranjan, and Robert W. Cottingham. 2018. Navigating the Maze: The Impact of Configurability in Bioinformatics Software. In *Proceedings of the 2018 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18), September 3–7, 2018, Montpellier, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3238147.3240466

1 INTRODUCTION

Bioinformatics software is becoming increasingly sophisticated and prevalent in its day-to-day use [7]. There is a plethora of software for aligning, assembling, and analyzing DNA sequences, or for optimizing and predicting growth in organisms using their metabolic networks. Databases have moved to common data formats and models, and high performance computing is allowing more scalable computation. As these bioinformatics systems become mainstream tools for the biology and bioinformatics community, their dependability and reliability becomes critical. Scientists are making decisions and drawing conclusions based on the software that they use. As in all empirical sciences, the results obtained need to be repeatable and explainable to others. Yet, recent research has suggested that better software engineering practices are needed in bioinformatics software [3, 14, 26, 27, 33, 43], as well as more sophisticated software testing techniques [21, 29].

Adding to this problem is that as the number and types of tools are increasing, they are also increasing in flexibility. For instance, the *Basic Local Alignment Search Tool* (BLAST) has multiple variants (e.g. web versus standalone, searching for proteins versus searching for nucleotides). The customization and flexibility of these tools indicates bioinformatics is entering a mature phase of software development — that of other highly-configurable software. Most tools now provide the end user with many customization options. The command line version of Nucleotide BLAST (BLASTn) has over 40 options that a user can modify when running a search. These options determine how an alignment search is conducted (e.g. what length subsequences to use in matching, or how the results are filtered based on quality thresholds). As has been well documented in the software engineering community, this leads to a broad spectrum of behaviors and choices for end users, yet it also creates some potential pitfalls.

Research in the software testing community has already extensively studied the impact that configurability has on testing and fault detection. Faults may be visible under only specific combinations of configuration options, and different code paths of a program are executed under different combinations of options [5, 30–32, 35, 36, 39–41, 46] which has led to many automated *configuration-aware* testing techniques.

While much of the literature on configurability from the software testing community focuses on interaction fault detection or performance issues [5, 18, 19, 24, 31, 35, 44, 45], configurability may lead to more subtle issues in a scientific domain because scientific software often uses heuristic or optimization approaches for solving problems. Some recent research has looked at variability in programs that calculate partial differential equations [10, 12] and for numerical solvers [42], but these are not in the bioinformatics

*Also with, Department of Plant Sciences, University of Tennessee.

ASE '18, September 3–7, 2018, Montpellier, France

ACM ISBN 978-1-4503-5937-5/18/09…$15.00
https://doi.org/10.1145/3238147.3240466

domain. Recent work that studies bugs in scientific software [8] does not address these issues of configurability, nor does other work on testing bioinformatics software [21, 29]. Morrison-Smith et al. examined how end users interact with bioinformatics software [33]. Users in their study were quoted as saying:

"*I don't necessarily know enough to make sure I'm picking our settings correct.*"

and

"At the end of the day, you just have faith that the program's working or not."

indicating that the user community needs support in understanding configurability and dependability for such systems. As researchers who have spent time working with multiple bioinformatics software programs, and who have experience with configurable systems, we have run into many of these problems ourselves. This has led us to ask what the impact of configurability is on this domain of software tools and what the implications might be on the research productivity of the broader community that uses these on a daily basis. We also want to understand if existing configuration-aware techniques can help improve this domain.

In this experience report we set out to answer these questions, and to provide insights for users and developers of bioinformatics software. While we focus on a single scientific domain, we believe these results may be more broadly applicable. We learn that configurability does impact bioinformatics software. More specifically (1) the configuration spaces are non-trivial to navigate and correctly model; (2) the functional outcomes, and those configuration options that control them are not always easy to identify; (3) performance may be impacted, sometimes significantly; and (4) configuration dependent faults exist.

The contributions of this paper are:

(1) A large case study demonstrating our experience with modifying configurations in three commonly used bioinformatics software systems across four different configuration models;
(2) Evidence that there is variability in both the functional and performance outcomes of these systems;
(3) A set of lessons learned and recommendations for developers and testers of these systems.

The rest of this paper is organized as follows. In the next section we provide some motivating examples. We follow that with a case study and results in Sections 3 and 4. We then present some discussion, lessons learned, and suggestions for improvement in Section 5. We end with conclusions and future work in Section 6.

2 MOTIVATING EXAMPLES

The bioinformatics software user community is broad, therefore developers should consider a spectrum of users. Fig. 1 shows the three primary users that we consider in this work. At the top level are the domain experts (or non-software experts). These users have a deep understanding of the biological problems they are trying to solve, but will unlikely be familiar with the configurable options and/or know which defaults are set in the software. Often, they work from a user interface that limits their options with respect to configurability. We examine one model in our study that is geared towards this group. As pointed out by Jin et al. [20], configurable systems usually have multiple ways to interact with configuration options,

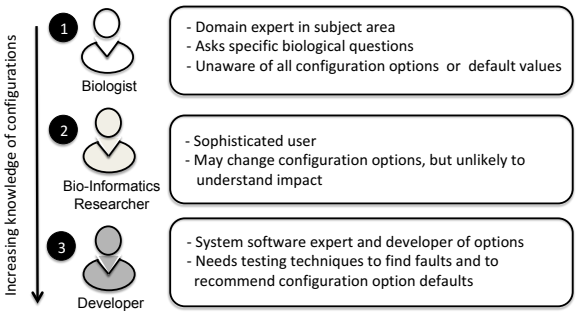

Figure 1: Three primary users.

and the highest level (such as menus or a graphical interface) often only provide a subset of those options.

The next type of user in this figure is the bioinformatics expert user who has both a deep knowledge of the biological domain, and with computation and automation. This type of user will often run experiments using scripts and will have some knowledge of which configuration options can be altered. However, they are unlikely to understand the full impact of those changes, nor do we expect them to be experts in software development or testing.

Finally, the expert developer is someone with extensive development experience with the domain, who has also built the application. They will be familiar with all of the configuration options and will have experimented with each, however, they may lack techniques and methods to test the large configuration space for correctness or to optimize and determine the best default values of each.

During the course of this experience report we have acted in the first two roles (both as biologist end users and as bioinformatics specialists). We have also reached out to developers with our findings and report on all of those experiences here.

We now present two motivating scenarios. These are based on recent interactions we have had with several popular bioinformatics software systems.

Scenario One We first describe our experience using an existing, popular, bioinformatics tool, *Basic Local Alignment Search Tool* (BLAST). Fig. 2 shows a set of configuration options for the online Nucleotide BLAST query system (BLASTn) [1] which searches a single (or multiple) database for the match to a single (or multiple) sequence (via queries). The user can simply run a query or they can open a screen that allows them to change the core search algorithm options. We show this screen in Fig. 2. Users can change any or all of these options. In this screen there are five pull down menus, five check boxes, and three text boxes that take numeric inputs, for a total of 13 configurable options. Each of the pull down menus has a number of values (or options) that can be selected.

Suppose we limit exploration to just four options for each of the pull down menus and text boxes. This leads to a possible space of (2^5 x 4^8) which is over two million configurations. We can reduce this slightly if one considers the dependency between the checkbox *species-specific repeats* (boxed in red in the figure) and its associated pull down, since that is only active when the box is checked, but this also complicates the modeling and exploration of the configuration space because that dependency must be satisfied.

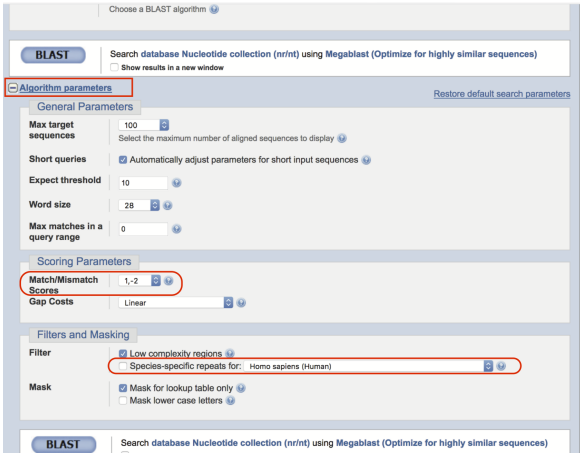

Figure 2: BLASTn online algorithm configuration options. Boxed elements are discussed in the text.

In this system each of the blue circled question marks provides user help. For instance, the *Match/Mismatch Scores* pull down list shows the following text when the question mark is clicked *'Reward and penalty for matching and mismatching bases. more...'*. If one clicks on *'more'* they are taken to a webpage with a long description of these configuration options. This is helpful to an expert user, but may confuse other users, and still leaves the bioinformatics expert with a less than desirable outcome. We also note that the command line version of this tool lacks the same level of help.

Even if the configuration options make sense to the bioinformatics expert, the actual practical impact on the returned sequences from the query when changing this option is unknown. Biologists often develop a *lab lore* of what to change and often leave the rest of the options alone. To actually understand the full impact, a biologist would need to set up a series of interactive experiments. Furthermore, this is just one configurable option. If we consider combinations with other choices, the number of configurations grows exponentially with the number of configurable options.

Scenario Two We next describe our experience with an application that assembles a set of short DNA strings called *reads* into larger continuous strings called *contigs* using an online web interface (i.e. again from a biologist's perspective). We attempted to recreate data from a tutorial for an application called MEGAHIT from the Department of Energy Systems Biological Knowledgebase (KBase) [2, 23]. The web version is simply a wrapper for a commonly used command line tool also called MEGAHIT [28].

We used the default values for each of the configuration options used in the tutorial. We noticed that one configuration option (k-max) did not make sense since it represents a read length (the number of elements in the sequence to read) and was set to 141, higher than the input sequence (100) used in the tutorial - typically something a user will not do. We ran the application anyway and got a completed assembly with 284 contigs. We then read the log and noticed that the system reported it changed the k-max for us to 119. But this was done silently (no notification) and was only found by our exploration of the log. This number is also greater than the maximum read size from the input set of reads. We then ran the tool and set the k-max to 119. This time we saw 285 contigs and thus inconsistent results. Last we set the k-max to 99 (below the read length) and expected to see one of the above numbers, but instead got 289 contigs.

We contacted a developer of the web app and were told that the problem is due to edge cases. The mismatch is due to the use of configuration options that combined with the input data do not make sense, and that the system autocorrects for us. However, the software did not document this clearly to the end user. Since the software was modifying values automatically to correct the incorrect k-max option value, we were getting inconsistent results. The developer confirmed that this is a bug and said that they would fix this in a future release.

These scenarios lead us to ask what the situation is like for the various users of bioinformatics software and what the impact of modifying configuration options might be. We do not expect broad expertise in software engineering from these users, therefore we want to understand what the implications might be for those using these systems. We also want to understand if configuration-aware techniques can help to uncover differences in either or both functional and performance outcomes of these tools and if there is a cautionary tale for this community. Our answer for both is yes.

3 CASE STUDY - EXPERIMENTING WITH CONFIGURATIONS

In this section we present our case study. We explore the following research questions:

RQ1 Does manipulating configurations in bioinformatics software lead to failures?

RQ2 What is the impact of manipulating configurations on program behavior? To answer this we ask two sub questions:

a) What is the impact on program functionality?

b) What is the impact on program performance?

Our last research question focuses on the ability to use common sampling techniques that might reduce user effort and improve scalability when exploring configuration options.

RQ3 How effective is sampling in these configuration spaces?

3.1 Bioinformatics Programs

We select three bioinformatics programs, all of which are configurable, and all available for different types of end users. These tools come from three different areas of study: genome annotation, genome assembly, and metabolic modeling. In the metabolic modeling tool we study two different versions. The first is meant for non-expert users and run from a graphical user interface (GUI). The second is geared towards the experienced bioinformatics user, and run on the command line. This gives us four unique configuration spaces. For each area, we chose a tool that (1) is widely used, (2) is configurable, (3) has some tutorial or other documentation that provides common inputs along with expected outputs to avoid author bias, and (4) can be run through automated scripts.

Table 1 shows the three bioinformatics program subjects along with the lines of code (LOC) using CLOC [6]. We describe each application below. We also show the total number of configuration options, along with those chosen for each model broken down by type. For example, the FBA-MFA subject has 21 Boolean and 14

float options. Last we give the size of the full configuration space including constraints (Section 3.2 contains details for each model).

Table 1: Case Study Subjects

	BLAST	MEGAHIT	FBA	
			GUI	MFA
LOC	1,095,106	38,093	28,909	28,909
Config. Opts.	58	7	17	52
Configuration Options Selected for Model				
Boolean	3	-	3	21
Int	-	4	-	-
Float	3	-	5	14
String	1	-	-	-
Total	7	48	8	35
Config. Space	3,000	260	25,000	1.28×10^{16}

1. BLAST Our first subject is Nucleotide BLAST *Basic Local Alignment Search Tool* (BLASTn), a popular bioinformatics tool developed by the National Library of Medicine [1].[1] Given an unknown DNA sequence (a query), the application attempts to align the sequence to a database of known DNA sequences in order to learn its likely function. Matches are returned with quality scores (percentage ID and e-values) that indicate how confident the search is with respect to each possible match. The queries can be run for either a single DNA sequence or a set of DNA sequences. We use as our input query, a file of 10,000 sequences from a yeast organism *Saccharomyces cerevisiae*, also known as *Baker's yeast* [37]. This input was used as an example in a tutorial on BLAST for bioinformatics researchers at our institution [13].

Our other subjects are the two most commonly used analysis tools taken from the open source GitHub repository of the U.S. Department of Energy's System Biology Knowledgebase (KBase) [2]. We can confirm results using their online web interface and are able to use their tutorials to obtain sample inputs/outputs along with descriptions of how each of the tools are used. At the time of this study, KBase has 145 apps deployed providing a wide range of functionality including next generation sequence analysis tools, DNA sequence assembly, annotation, and modeling of metabolic pathways. KBase tools have been used in over 37 publications, and the methods implemented in the tools in thousands of publications.
2. MEGAHIT Our next subject is the DNA assembler *MEGAHIT* [22]. The KBase tool is a wrapper for an existing version of MEGAHIT [28]. When DNA is sequenced the result is a set of small sequences of base-pairs called *reads*. In order to combine these small sequences, assembly is used to connect them together into longer sequences called *contigs*. This has been used 1,120 times inside of the KBase narrative as of April, 2018. We use the default paired-end-library from KBase which contains 386,106 reads all of length 100 basepairs (bp) from the *Rhodobacter* bacterium. This library is used in the KBase online tutorial [23].
3 and 4. Flux Balance Analysis The last program is a metabolic modeling toolkit that performs a flux balance analysis (FBA). The metabolism of an organism can be modeled as a set of concurrent chemical reactions (a reaction network). FBA optimizes reaction fluxes (flows) through an organism's metabolic reaction network to predict the organism's growth [17]. It uses linear programming as the underlying optimization technique. We utilize the open source standalone version (toolkit) of the FBA from the KBase GitHub software distribution kit [16]. KBase has a web version of the FBA application (we refer to this as the FBA-GUI model) that has been run 7,803 times as of April, 2018. The standalone toolkit (we call this the FBA-MFA) has a larger set of configurable options available to the user. A subset of these options are available within the online version in KBase. To understand if the web narrative provides similar behavior with respect to configurability, we model these independently. We fix the organism (*Escherichia coli*) and growth medium (Carbon-D-Glucose) to match that of the online tutorial which describes how to use FBA in KBase.

3.2 Creating the Models

Our first step is to develop configuration models for each application (we provide these on the supplementary website).[2] We use available documentation, such as tutorials, the application interface, online documentation, and we hold discussions with domain experts. Since we are evaluating functionality as one objective, we try to avoid modifying configuration options that are clearly functional, therefore we leave those out of our models. For example, we do not change the growth media in FBA as that will clearly impact how the organism grows. Other configuration options are removed for reasons including: dependencies with default configuration options, they only impact output formatting, filter/threshold, and unique string values that could not be put into equivalence classes. In choosing the values for the configuration options we always include the default option and then try to evenly distribute equivalence classes for other values when it makes sense. We rely heavily on guidance from the documentation (or online developer forums) to determine values.

We found that it was non-trivial to construct a configuration model for each subject. Some configuration options are ill-defined, there is a lack of documentation, or there are conflicting summaries in multiple help documents. We also found that the explicit ranges of values are often missing and we had to reverse engineer default values. We were quickly lost in the maze of configuration options. In order to find our way out we communicated with developers and expert users of the tools to identify configurations of interest for each model. We discuss some of these experiences next.
BLAST We began with the full list of 58 command line configuration options from the user manual and scheduled two consultations with an expert user (who is not one of the authors of this paper). The user was only familiar with some of the configuration options, therefore their input was useful, but we still ran into several challenges. We first removed any clearly functional, string, and set configuration options giving us an initial model of 18 configuration options. However, in running preliminary experiments, we ran through a total of five iterations of the model due to errors and inconsistencies in the documentation. At each step we had to stop and trace the errors back to their cause. Due to space constraints

[1] The BLAST software contains different variants. Table 1 shows the LOC for the full BLAST software system. We utilize only one variant of BLAST (BLASTn) in this work.

[2] supplementary dataset:
https://github.com/mikacashman/ASE18SupResources

we do not list them all, but summarize the key learning points. In preliminary testing we came across one error:

> *BLAST query/options error: Greedy extension must be used if gap existence and extension options are zero. Please refer to the BLAST+ user manual.*

This error message is due to incompatible configuration options, but does not name them. After searching web forums with little success, we searched the source code and found that the *no_greedy* option had constraints. The configuration option *no_greedy* can not be used when *gapopen* and *gapextend* are zero. In our model we were not changing these values because they are functional, and their default is 0, therefore we are unable to include the *no_greedy* option in our model either.

Table 2: BLAST Configuration Model

Option Name	Type	Values
dust	String	yes, "20 64 1", no
soft_masking	Boolean	True, False
lcase_masking	Boolean	True, False
xdrop_ungap	Real	0, 0.1, 0.5, 20, 100
xdrop_gap	Real	0, 0.1, 0.5, 30, 100
xdrop_gap_final	Real	0, 0.1, 0.5, 10, 100
ungapped	Boolean	True, False

Table 3: MEGAHIT Configuration Model

Option Name	Integer Range	Values
min-count	[2,10]	2, 4, 6, 8, 10
k-min	[1,127] odd only	15, 21, 59, 99, 141
k-max	[1,255] odd only	15, 21, 59, 99, 141
k-step	[2,28] even only	2, 6, 12, 20, 28

Constraints: k-min < k-max, k-step < k-max–k-min

Table 4: FBA Configuration Models

Option Name	Type	Values
flux variability analysis	Boolean	True, False
minimize flux	Boolean	True, False
simulate all single KOs	Boolean	True, False
MaxC	Float	0, 25, 50, 75, 100
MaxN	Float	0, 25, 50, 75, 100
MaxO	Float	0, 25, 50, 75, 100
MaxP	Float	0, 25, 50, 75, 100
MaxS	Float	0, 25, 50, 75, 100
Example of Additional MFA Options		
maxDrain	Float	0, 250, 500, 750, 1000
minDrain	Float	0, -250, -500, -750, -1000
deltaGSlack	Float	0, 5, 10, 15, 20
find_min_media	Boolean	True, False
allRevMFA	Boolean	True, False
useVarDrainFlux	Boolean	True, False

We also discovered configuration options that behave as filters that were not obvious when studying the user manuals. Identifying such functional options was a hard task. Even after looking though two user manuals and consulting two users some configuration options' functionality were still not clear. The *word_size* option was removed as it was discovered to be a filtering configuration option and would clearly change the functional outcome of the query. We discuss the importance of this in Section 5.

One modification to the model was triggered by a sanity check we performed to confirm the default configuration reported in the manuals was in fact the default. We compared the configuration option manually set (*blastn -db [DATABASE] -query [QUERY] -out [OUT] -word_size 11 -dust "20 64 1" -soft_masking true -xdrop_ungap 20 -xdrop_gap 30 -xdrop_gap_final 100 -window_size 40 -off_diagonal_range 0*). To our surprise, the results were not the same. We first referred back to the manual and command line help menu to ensure the default values of the configuration options were correct (they were). After continued inspection we noticed an implicit configuration option (called *task*) that did not have its own declaration in the manual or the help menu. BLASTn can be used with four different *tasks* and we use the default. However, the default value listed for a BLASTn option we do use (*window_size*) is for a different task, and the task we use has no default value listed for *window_size*. So we removed *window_size* from our model to avoid this constraint. There are no constraints in the final model.

MEGAHIT. There are seven configuration options accessible from the web interface. Many of these options determine a list of integer values called the k-list. This controls the length of the sub-string used in the database search. For example, we can set a *k-min*, *k-max*, and *k-step* or explicitly set the entire list with the *k-list* configuration option. Similarly the option *parameter preset* is a different method of fixing the same options — we exclude both from the model because they impact functionality. We also fix the configuration option *min-contig-length* because it directly impacts our functional result. We confirmed this model with a KBase developer. When working with this model we ran into several issues of consistency which we discussed in Section 2.

The MEGAHIT model has constraints between the configuration options (such as k-min < k-max). This constraint was not listed in the documentation, but is considered common knowledge (min should be less than max) and was mentioned to us by the developer. We also confirmed this constraint by running initial experiments where we received an error message. Although it properly returned an error, the message was not descriptive enough for an end-user to understand. The final MEGAHIT model is in Table 3 and has 4 configuration options.

FBA For the FBA-GUI subject we first fixed all of the input files (describing the growth conditions) and ruled out configuration options that were of type *set*. Set options were removed as the number of configuration options is on the order of 10^{258} with no logical method of partitioning. We also do not use an optional input (*expression data set*) to keep the experiment simple. There was one float type configuration option that was not included in the command line version of the tool, so it was excluded from the model as well to ensure we could compare the two. Table 4 displays the configuration options for the FBA models. The FBA-GUI model

consists of the first eight configuration options, three Boolean, and five floats evenly partitioned into five values.

The FBA-MFA model contains 27 additional advanced configuration options, five of which are displayed in the table. The complete FBA-MFA model contains 35 configuration options, 21 Boolean and 14 floats evenly partitioned into five values. There are no constraints in either model. We believe that the lack of constraints may be partially due to our selection of configuration options, and leave further exploration of this as future work.

3.3 Measures of Variability

For program functionality we choose common use cases for each tool. For performance we measure the program execution time (in seconds). We describe each of the functional metrics next.

In BLAST the input is a set of query sequences of unknown function that are checked against a database of query sequences. There are two main use cases: (1) the user inputs a *set* of query sequences and then looks at the best quality hits (defined as 100% basepair matches and an e-value of 0.0) or (2) the user inputs only one query sequence at a time and checks for any hits against that sequence. We explore both use cases. In the first, since there are large numbers of hits returned, we look at the Top 5, 10, 20, and 100 highest quality hits and compare them to the Default Top 5, 10, 20, 100. We keep the number small since the analysis of the returned hits is often a manual effort by the user. For the second use case we count the *number of hits* for each individual query sequence.

MEGAHIT is used to assemble short DNA reads from sequencers into continuous DNA sequences. The functional dependent variable is the *number of contigs* generated. This represents how many continuous reads of DNA the algorithm was able to assemble. The objective of the tool is to get the smallest number of long contigs.

The result of the FBA analysis is called the *Objective Value* (OV) in our tool. We use this as the functional output.

3.4 Experimental Setup

The BLAST model has a configuration space of 3,000 which is small enough for us to enumerate and evaluate. We exhaustively run all configurations for our experiments. We run the command line *BLASTn* version 2.6.0 on a LINUX cluster with 100 jobs in parallel, each job on 1 node with 5GB memory.

The configuration model for MEGAHIT has 260 configurations. We exhaustively run all configurations in this space as well. The MEGAHIT app is version used is 2.2.8 in KBase. We run these experiments in parallel on 10 virtual machines. Each machine runs CentOS-7-x86_64 with 80 GB disk and 2MB RAM.

The FBA subjects' source code is extracted and run on a LINUX cluster in order to scale the experiments and not overload the KBase servers. Since the total configuration space for FBA-MFA is too large to study (1.28×10^{16}) we randomly selected 125,000 of these combinations. We use FBA version 1.7.4 running 1,000 jobs in parallel, each job on 1 node with 10GB memory. In order to test all configurations in the FBA-GUI model (25,000) we test it directly from the MFAToolkit also running 1,000 jobs in parallel. We also test and confirm that our functional results matched on a sample of these when running manually in the KBase online narrative. In early iterations of our study on FBA-MFA, some of our runs were hanging (not completing overnight) and this caused the experiments to stop. We added a timeout factor of 1,000 seconds to prevent this problem and kill any job that goes beyond this time. We note that we did have some runs of exactly 1,000 seconds (seen in our data) that did complete and produce valid outputs. We do not explore the best cutoff time further in this work, but believe these are outliers.

3.5 Sampling

To evaluate RQ3 we applied a common configuration-aware testing technique, combinatorial interaction testing (CIT) [4, 30, 38, 39]. CIT combines all t-way combinations of configuration options together in at least one run of the sample. Research in traditional software testing suggests that t between 2 and 6 should be sufficient and that 2 or 3 finds most faults [25]. We used strength 2 to 6 in our study. We used the CASA tool [9] for our lower strength samples, but were unable to build some of the higher strength samples so used the ACTS [47] tool for those. When using CASA we generated 30 samples since it uses randomness and each sample is different. With ACTS the generation was deterministic therefore we have a single CIT sample. We also applied a variant of option-wise testing [44] which has been used in configuration performance tuning. Option-wise sampling implies binary configuration options and tests each configuration option being enabled once while the remaining configuration options are all disabled. Since the configuration options in our models are not all binary, we instead use the default configuration setting as our baseline. Then for each configuration option and for each value of the configuration option, we change that one option in the default configuration.

4 RESULTS

In this section we present the results for each of our research questions. We discuss the implications in Section 5.

4.1 RQ1: Failure Detection

BLAST We went through several iterations of the BLAST model (discussed in Section 3.2) in which we ran into many error messages. However these turned out to be due to incorrect configurations and violated dependencies. Once we fixed these, we did not find any other errors while running the 3,000 configurations.

MEGAHIT 23 configurations failed (9.62%). Five configurations reported an "*Error occurs when running sdbg builder count/read2sdbg*" and 18 configurations reported the error "*Error occurs when assembling contigs for k = 99*" error. We contacted the developers and were told that both cases are valid configurations, but incompatible with the input data. The first is due to setting two configuration options (k-min and k-max) both to 141. The second error only occurs when k-min=99 and minCount>4. In these cases the program should have resulted in no contigs, but produced an error. We note that we worked with bioinformatics experts who use this tool to model our configuration space, therefore these are not obvious edge cases. The developer explained that these are both cases where there were no assembled reads and the tool was failing ungracefully. We did see 35 more cases where there was no assembly, but these did not fail so we consider them in RQ2. In total 237 out of 260 configuration ran without errors.

FBA-MFA Of the 125,000 configurations tested, 1,285 (1.03%) of these configurations failed. We see five distinct error messages (Table 5 summarizes these). 631 runs timed out after 1,000 seconds. 44 configurations caused an error in the linear programming solver (called SCIP). 54 tests reported an objective value (OV) of *negative 0*, however the smallest OV should be 0. A total of 447 runs aborted, and 27 were segmentation faults.

Table 5: FBA-MFA - Errors

Total runs	**125,000**
Timeout	631 (0.50%)
SCIP Error	44 (0.04%)
Negative 0 OV	154 (0.12%)
Aborted	447 (0.36%)
Segmentation Fault	27 (0.02%)
Total w/out Errors	**123,715 (98.97%)**

We first re-ran these tests to confirm that the failures were deterministic and then we reported some of these on the KBase help forum and communicated directly with developers of the FBA program. Several of the errors were confirmed. Some were unsurprising (again due to nonsensical combinations of configuration options) but some were also tagged as real bugs that should be fixed via patches to the program. In response to the SCIP error, the help board told us it was due to the linear programming solver, not the FBA. We got slightly different feedback from the developer who was confused by the error. The resulting OV value it gives us when this occurs is 0 (no growth) so the functional result is correct. The Aborted error is a memory allocation issue in the C++ code. We are continuing dialog with the developer to investigate this further. We discuss more developer feedback in Section 5.

FBA-GUI Of the 25,000 combinations, 16 of these combinations resulted in an error. All were the SCIP error (similar to the FBA-MFA example). We confirmed some of these within the KBase online GUI to ensure that the error occurs there as well. All again returned the correct zero OV (no growth), but there was an error reported in the online log. In this case the error may not be obvious to the user, because the GUI handles this gracefully and returns a result. The user must examine the log explicitly to see the error.

Summary of RQ1. Three of the four applications suffered from failures due to us using invalid combinations of configuration options (i.e. missed dependencies). We also saw some errors that are due to either external libraries (linear solvers) or the bioinformatics application itself.

4.2 RQ2 (a): Functionality

BLAST We examined two different use cases for BLAST as discussed in Section 3.3. For Use Case (1) Table 6 displays the count of Top X hits that match the default settings exactly (in terms of sequence ID and order in in list). For example, the count for Top 5 means that 2,000 of the 3,000 configurations' Top 5 hits match the Top 5 hits from the default configuration. Overall we see that ⅔ of the configurations do not vary in the Top 20 hits. The remaining 1,000 have a different list of hits than the default. If we look at the Top 100 hits only 640 configurations gave the same result as the default options.

Table 6: BLAST Functional Variance for Use Case 1

	Count (percent)
# hits match Top 5	2,000 (66.67%)
# hits match Top 10	2,000 (66.67%)
# hits match Top 20	2,000 (66.67%)
# hits match Top 100	640 (21.33%)
Total in sample	**3,000**

For Use Case (2) we look at all 10,000 input sequences as individual test cases (Table 7). We see that over half (53.69%) never find a hit across all 3,000 configurations which is typical behavior. Another 17.60% of the query sequences always have 1 or 3 hits (no variation). The remaining 2,871 (28.71%) of the query sequences have a varying number of hits across the configuration options we explored.

Table 7: BLAST Functional Variance for Use Case 2

Total # Query Sequences	**10,000**
Sequences w/out variance (0 hits)	5,369 (53.69%)
Sequences w/out variance (1-3 hits)	1,760 (17.60%)
Sequences w/ variable hits	**2,871 (28.71%)**

MEGAHIT Of the 260 configurations, we removed the 23 that resulted in an error. Another 35 (14.50%) resulted in zero assembled contigs and produced no output. Therefore the total number of configurations that gave valid assemblies was 202. In Fig. 3 we show the number of contigs for the 202 configurations with a valid assembly (of at least 1). We see a large variation in the number of contigs. The median number of contigs is 324, but the variation range is from 1 to 672 contigs. This means that configurations explored have a large impact on the functional result of MEGAHIT.

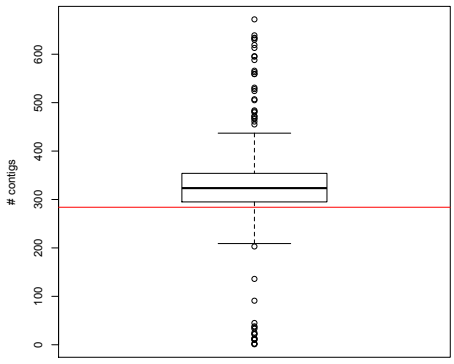

Figure 3: MEGAHIT - The red horizontal line represents the default number of contigs at 284.

FBA-MFA We removed the 1,285 error configurations from the functional analysis. Of the remaining 123,715 configurations we saw 39 unique objective values. This suggests that the configuration

options we are changing have a large impact on functionality. The majority of the combinations (96.13%) do not grow resulting in an OV of zero. This is interesting because we fixed the input model and growth media to match that of the online tutorial using all default values and that always grows (0.507834). Fig. 4 shows a distribution of OVs with the `x-axis` representing OVs, and the `y-axis` representing the count. Note that the left most bar (a zero OV) is elided to fit into this graph. We have identified the default growth on the graph. It turns out that this is not the most frequently observed (positive) OV. But it is second (705 occurrences) to a slightly higher OV that occurs 978 times. This suggests that the default values are not optimal for this very common organism.

FBA-GUI For these 25,000 tests we saw four unique objective values (Table 8). Compared with the FBA-MFA tests this has less variation. However, the majority (67%) still do not grow. Only 18.40% match the online tutorial with default values, and interestingly, we can not obtain the higher OV values using the GUI variant of this application suggesting that the configuration options of the command line tool provide additional functional benefit to this application (or they are modifying the "fixed" media indirectly).

Table 8: FBA-GUI - Functional Variance

OV	Frequency
0	16,808 (67.23%)
0.395492	2,048 (8.19%)
0.423195	1,536 (6.14%)
0.507834	4,608 (18.43%)

Summary of RQ2 (a). We saw functional variability in all of our subjects. The implication for reporting scientific results may be significant. The greatest variability was in MEGAHIT signaling the configuration options in the model may be related to functionality.

4.3 RQ2 (b): Performance

BLAST For BLAST we did not see much variation in the runtime. Most configurations run in 6 seconds (2,959 test cases) and the remaining 41 tests run in 11 seconds.

MEGAHIT We see a large variation in the runtime of the tests that produce assemblies. Fig. 5 shows a boxplot with a median runtime of 313 seconds and a range from 145 to 1,148 seconds. All tests that lead to an error run in 39 seconds.

FBA-MFA The runtime varies by configuration from less than 1 second to the timeout of 1,000 seconds. 99.50% of the tests finish in under 100 seconds. We also see a variance in runtime within the same OV shown in Fig. 6 with the `x-axis` representing the OV values, and the `y-axis` representing the runtime in seconds. Most of the OV values have a median runtime between 6 and 40 seconds, but there are many cases of outliers that take significantly longer to run. This tells us we can achieve the same objective value in some cases in 1 or 1,000 seconds. We also confirmed these runtimes are consistent. We randomly picked four OVs and ran all of their combinations (562 in total) 10 times each. There was a low variance across each the 10 runs (median of 2.5 seconds).

FBA-GUI All of the configurations finished in less than 42 seconds with the majority (99.28%) finishing in under 12 seconds. We see the least amount of variability when the OV is zero (the model does not grow). However it can still fail to grow but, take up to 21 seconds. The non-zero OVs all have a median runtime of 6 seconds, but can range up to 41 seconds. Similar to the FBA-MFA model, we see we can achieve the same OV with different runtimes as seen in Fig. 7.

Summary of RQ2 (b). For two of the applications, the performance was not impacted by modifying configurations. However in FBA the performance varies quite a bit by configuration and it is possible to get the same OV with very different runtimes.

4.4 RQ3: Sampling

For this RQ we wanted to understand if some common sampling techniques can be used for efficiency. We leave a full analysis as future work. We only present data from the two FBA tools since we could not exhaustively test those configuration spaces.

FBA-MFA Results can be seen in Table 9. For functional output we see that a 2-way CIT sample only produces three unique objective values compared to the 39 we saw in our large (125,000) sample. We don't see at least half of these unique OVs until we scale up to a 5-way CIT sample. Similarly, if we analyze the samples for errors we don't consistently see all five errors, until the 4-way CIT sample. This suggests that the software has higher order interactions. The option-wise testing method only captured five unique OVs and only one of the errors.

FBA-GUI Results can be seen in Table 10. The total number of unique OV values (4) in all the samples, show that the CIT samples can sample the GUI functionality. The SCIP error doesn't show up consistently (in at least half the runs) until the 5-way CIT sample.

Summary of RQ3. The 5-way CIT samples detected functional variance and all errors using only a fraction of the configuration space, however the lower strength CIT was not as effective. Option-wise testing did not prove effective in this case. The results do suggest there is potential for sampling to help improve efficiency.

5 DISCUSSION AND LESSONS LEARNED

Further Developer Feedback One of the developers of the FBA tools was very responsive to our information sharing and confirmed that this type of analysis would be useful for developers. He confirmed that many of these errors are due to real faults in the system. Some were unsurprising to him because they were due to combinations of configuration options that were never meant to be run together, some others were puzzling and are being examined now to find the root cause for a bug patch. We believe the first set of errors can be fixed by either preventing their combination from happening via code, or through better documentation. The second class of errors will lead to program repair - both valuable additions.

He provided us additional feedback including:

- It was surprising to see some particular configurations result in a positive OV value meaning the organism grew. According to his expertise these should not have grown signaling a possible bug.
- He was surprised that some cases of a zero OV (no growth) took a significant time to run, suggesting cases of no growth are not currently optimized in the code.

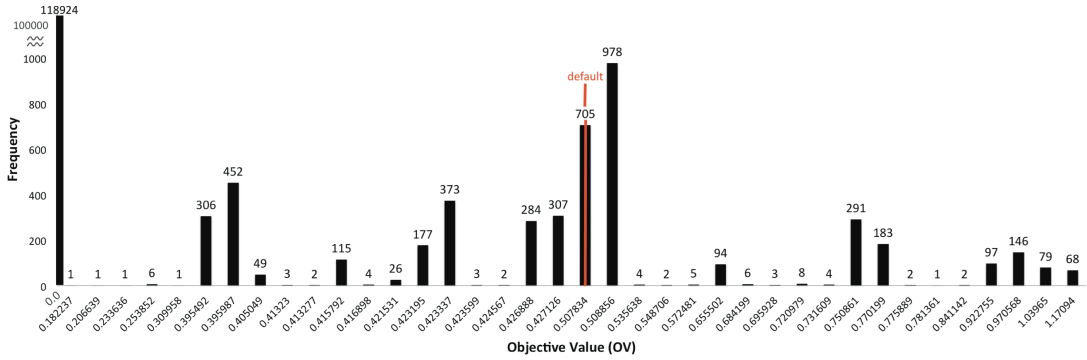

Figure 4: MFA - Boxplot of the frequency of OVs returned from the FBA analysis.

Table 9: FBA-MFA Sampling Results

	2-way CIT (30 runs)	3-way CIT (30 runs)	4-way CIT (1 run)	5-way CIT (1 run)	6-way CIT (1 run)	Option-Wise (1 run)	Random (1 run)
Average # in sample	43	324	2,423	14,346	79,372	78	125,000
Average # uniq OVs	3	10	19	29	40	5	39
Errors (times seen in total # of runs)							
Timeout	4	24	1	1	1	0	1
SCIP	0	9	1	1	1	1	1
Aborted	4	15	1	1	1	0	1
Segmentation	0	7	1	1	1	0	1
Negative 0	3	16	1	1	1	0	1

Table 10: FBA-GUI Sampling Results

	2-way CIT (30 runs)	3-way CIT (30 runs)	4-way CIT (30 runs)	5-way CIT (1 run)	6-way CIT (1 run)	Option-Wise (1 run)	Total Configs
Average # in sample	25	152	6,759	3,129	6,250	24	25,000
Average # uniq OVs	4	4	4	4	4	4	4
Errors (times seen in total # of runs)							
SCIP	0	2	14	1	1	0	1

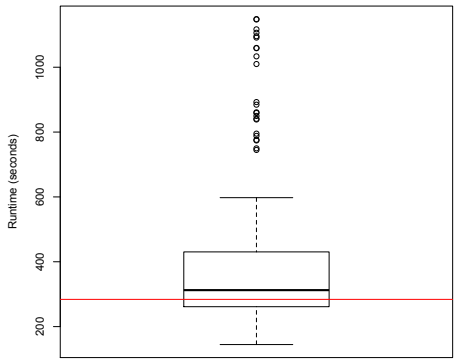

Figure 5: MEGAHIT - Runtimes. The red horizontal line represents the default runtime of 284 seconds.

- He remarked that the SCIP error was both "interesting" and "confusing" and asked for more analysis to help them locate the bug(s).

In summary, the developer was able to (1) learn about unknown errors and (2) learn about new ways to potentially optimize runtime. He also asked us for more details of our analysis and is actively working on repairing these issues. Finally, he asked us to run additional experiments with his models to explore some additional configuration options.

For MEGAHIT, the developer plans to update the documentation due to our feedback which may help to improve user understanding and experience of that tool.

5.1 Lessons Learned

We learned several valuable lessons about configurable bioinformatics software, some of which we believe can be generalized to other scientific software.

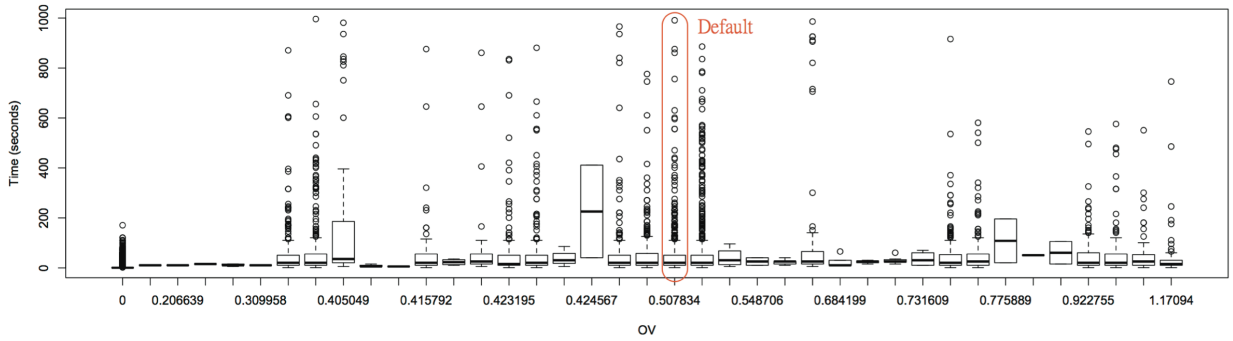

Figure 6: FBA-MFA - Boxplot runtime per OV

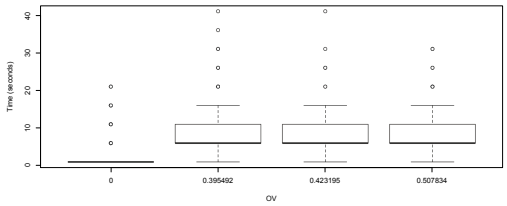

Figure 7: FBA-GUI - Boxplot runtime per OV.

(1) The meaning of configuration options and constraints between options are often not clearly documented or easy to extract even by experienced users.
(2) Changing configuration options can have a large effect on functionality and it is hard to determine from documentation which configuration options matter.
(3) Determining an exact functional output (oracle) for the problem was not straight forward. This suggests that the results of some experiments using these tools can be left open to interpretation.
(4) The same functionality can be achieved with varying performance.
(5) Unknown and unexpected errors exist in these systems under specific configurations of parameters.
(6) Sampling may be an effective tool for testing these systems, but needs more exploration.

5.2 Suggestions for Improvement

(1) **Create clear distinctions between functional and performance parameters.** Despite our effort to exclude functional options we were unsuccessful. We suggest developers divide configuration options into classes by their intended impact on functionality and document this well.
(2) **Provide automated and transparent methods for handling dependencies.** We found dependencies that caused errors, and one tool auto corrected to fix dependencies. Both caused problems. We believe that developers should provide automated techniques to avoid incompatible options, but that the dependencies should also be clearly explained to the end user.
(3) **Provide automated tuning algorithms.** Despite our combined expertise with these tools and our discussions with expert developers, we found that there was no explanation for some of the behavior we observed. There is an opportunity for developers to create automated tuning algorithms to guide all levels of users when manipulating and exploring configuration spaces.
(4) **Developers can benefit from automated configuration-aware testing techniques.** Our experiments suggest that developers can utilize existing automated configuration-aware testing tools to build more dependable software. Not only can it help them find latent faults, but it can help to understand the differences in functional and performance variance.

6 CONCLUSIONS AND FUTURE WORK

Bioinformatics software is increasingly being used by different types of users and has become highly configurable. We have studied this issue to learn how we can build automated techniques for these tools. We show via a case study that manipulating configuration options in three popular bioinformatics programs can lead to lots of variability. We find the functionality of these systems are dependent on the chosen configuration options. We also find that understanding the configuration models and underlying constraints is non-trivial and that developers can benefit from configuration-aware testing tools. Last we see that common existing sampling techniques may be beneficial. We have provided a set of actions that developers can take to improve the repeatability and dependability of these systems.

In future work we plan to build automated approaches to help address the problems that we found. We may leverage existing frameworks such as SPL Conquerer [44] which has already been used in the scientific domain [11] and will explore domain specific languages to help the users. We also plan to expand our models to incorporate more complex constraints. Doing so may require adapting work from [15]. Finally, our approach to creating the configuration models was largely manual. We plan to automate this process as seen in other work [34].

ACKNOWLEDGMENTS

This work is supported in part by NSF grant CCF-1745775 and by the Office of Biological and Environmental Research's Genomic Science program within the U.S. Department of Energy Office of Science, award number DE-AC05-00OR22725.

REFERENCES

[1] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. 2018. Basic local alignment search tool. *Journal of Molecular Biology* 215 (2018). Issue 3. https://blast.ncbi.nlm.nih.gov/.

[2] Adam P Arkin, Robert W Cottingham, Christopher S Henry, Nomi L Harris, Rick L Stevens, Sergei Maslov, et al. 2018. KBase: The United States Department of Energy Systems Biology Knowledgebase. *Nature Biotechnology* 36 (2018), 566–569.

[3] Parmit K. Chilana, Carole L. Palmer, and Andrew J. Ko. 2009. Comparing bioinformatics software development by computer scientists and biologists: An exploratory study. In *Proceedings of the ICSE Workshop on Software Engineering for Computational Science and Engineering (SECSE)*. IEEE Computer Society, 72–79.

[4] David M. Cohen, Siddhartha R. Dalal, Jesse Parelius, and Gardner C. Patton. 1996. The combinatorial design approach to automatic test generation. *IEEE Software* 13, 5 (1996), 83–88.

[5] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. 2008. Constructing Interaction Test Suites for Highly-Configurable Systems in the Presence of Constraints: A Greedy Approach. *IEEE Transactions on Software Engineering* 34, 5 (2008), 633–650.

[6] Al Danial. 2015. CLOC - Count Lines of Code. website. http://cloc.sourceforge.net/

[7] Geraint Duck, Goran Nenadic, Michele Filannino, Andy Brass, David L. Robertson, and Robert Stevens. 2016. A survey of bioinformatics database and software usage through mining the literature. *PLoS ONE* 11, 6 (2016).

[8] Anthony Di Franco, Hui Guo, and Cindy Rubio-González. 2017. A Comprehensive Study of Real-World Numerical Bug Characteristics. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE.

[9] Brady J. Garvin, Myra B. Cohen, and Matthew B. Dwyer. 2010. Evaluating improvements to a meta-heuristic search for constrained interaction testing. In *Empirical Software Engineering (EMSE)*, Vol. 16. Springer US, 61–102. Issue 1.

[10] Alexander Grebhahn, Christian Engwer, Matthias Bolten, and Sven Apel. 2017. Variability of stencil computations for porous media. In *Concurrency and Computation: Practice and Experience*, Vol. 29. John Wiley & Sons, Ltd. Issue 17.

[11] Alexander Grebhahn, Sebastian Kuckuk, Christian Schmitt, Harald Köstler, Norbert Siegmund, Sven Apel, Frank Hannig, and Jürgen Teich. 2014. Experiments on optimizing the performance of stencil codes with SPL Conqueror. In *Parallel Processing Letters*, Vol. 24. Issue 3.

[12] Alexander Grebhahn, Carmen Rodrigo, Norbert Siegmund, Francisco J Gaspar, and Sven Apel. 2017. Performance-influence models of multigrid methods: A case study on triangular grids. In *Concurrency and Computation: Practice and Experience*, Vol. 29. John Wiley & Sons, Ltd. Issue 17.

[13] HCC. 2017. Using BLAST on HCC. https://github.com/unlhcc/job-examples

[14] Dustin Heaton and Jeffrey C. Carver. 2015. Claims About the Use of Software Engineering Practices in Science. *Inf. Softw. Technol.* 67, C (Nov. 2015), 207–219.

[15] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. 2015. Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines. *IEEE/ACM International Conference on Software Engineering (ICSE)* 1 (2015), 517–528.

[16] Christopher S. Henry. 2017. MFAToolkit GitHub Repository. https://github.com/cshenry/fba_tools/tree/master/MFAToolkit

[17] Christopher S. Henry, Matthew DeJongh, Aaron A. Best, Paul M. Frybarger, Ben Linsay, and Rick L. Sevens. 2010. High-throughput generation, optimization and analysis of genome-scale metabolic models. *Nature Biotechnology* 28, 9 (2010), 977–982.

[18] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 497–508.

[19] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. 2017. Transfer Learning for Improving Model Predictions in Highly Configurable Software. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 31–41.

[20] Dongpu Jin, Xiao Qu, Myra B. Cohen, and Brian Robinson. 2014. Configurations Everywhere: Implications for Testing and Debugging in Practice. In *International Conference on Software Engineering, Software in Practice Track (ICSE)*. ACM, 215–225.

[21] Amir Hossein Kamali, Eleni Giannoulatou, Tsong Yueh Chen, Michael A. Charleston, Alistair L. McEwan, and Joshua W.K. Ho. 2015. How to test bioinformatics software?. In *Biophysical Reviews*, Vol. 7. Springer Berlin Heidelberg, 343–352. Issue 3.

[22] kb-megahit 2017. KBase MEGAHIT SDK Repository. https://github.com/kbaseapps/kb_megahit

[23] kbase-assembly 2017. Microbial Genome Assembly and Annotation Tutorial. https://narrative.kbase.us/narrative/notebooks/ws.18188.obj.6

[24] Chang Hwan Peter Kim, Darko Marinov, Sarfraz Khurshid, Don Batory, Sabrina Souto, Paulo Barros, and Marcelo d'Amorim. 2013. SPLat: Lightweight dynamic analysis for reducing combinatorics in testing configurable systems. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on Foundations of Software Engineering (ESEC/FSE)*. ACM, 257–267.

[25] D. Richard Kuhn, Dolores R Wallace, and Albert M. Gallo. 2004. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering* 30, 6 (2004), 418–421.

[26] Brendan Lawlor and Paul Walsh. 2015. Engineering bioinformatics: Building reliability, performance and productivity into bioinformatics software. *Bioengineered* 6, 4 (2015), 193–203.

[27] Felipe da Veiga Leprevost, Valmir C. Barbosa, Eduardo L. Francisco, Yasset Perez-Riverol, and Paulo C. Carvalho. 2014. On best practices in the development of bioinformatics software. *Frontiers in Genetics* 5 (2014), 199.

[28] Dinghua Li, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak-Wah Lam. 2015. MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinformatics* 31, 10 (2015).

[29] Anders Lundgren and Upulee Kanewala. 2016. Experiences of testing bioinformatics programs for detecting subtle faults. In *Proceedings of the International Workshop on Software Engineering for Science - SE4Science*. IEEE, 16–22.

[30] Flávio Medeiros, Christian Kästner, Márcio Ribeiro, Rohit Gheyi, and Sven Apel. 2016. A Comparison of 10 Sampling Algorithms for Configurable Systems. In *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 643–654.

[31] Flávio Medeiros, Iran Rodrigues, Márcio Ribeiro, Leopoldo Teixeira, and Rohit Gheyi. 2015. An empirical study on configuration-related issues: Investigating undeclared and unused identifiers. In *International Conference on Generative Programming and Component Engineering (GPCE)*. ACM, 35–44.

[32] Jens Meinicke, Chu-Pan Wong, Christian Kästner, Thomas Thüm, and Gunter Saake. 2016. On Essential Configuration Complexity: Measuring Interactions In Highly-Configurable Systems. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 483–494.

[33] Sarah Morrison-Smith, Christina Boucher, Andrea Bunt, and Jaime Ruiz. 2015. Elucidating the role and use of bioinformatics software in life science research. In *Proceedings of the British HCI Conference*. ACM, 230–238.

[34] Sarah Nadi, Thorsten Berger, Christian Kästner, and Krzysztof Czarnecki. 2015. Where do configuration constraints stem from? An extraction approach and an empirical study. *IEEE Transactions on Software Engineering (TSE)* 41 (March 2015), 820–841. Issue 8.

[35] ThanhVu Nguyen, Ugur Koc, Javran Cheng, Jeffrey S. Foster, and Adam A. Porter. 2016. iGen: Dynamic Interaction Inference for Configurable Software. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. ACM, 655–665.

[36] Marius Nita and David Notkin. 2009. White-box approaches for improved testing and analysis of configurable software systems. In *International Conference on Software Engineering (ICSE)*. IEEE, 307–310.

[37] U.S. National Library of Medicine. [n. d.]. National Center for Biotechnology Information.

[38] Justyna Petke, Myra B. Cohen, Mark Harman, and Shin Yoo. 2013. Efficiency and early fault detection with lower and higher strength combinatorial interaction testing. In *European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. ACM, 26–36.

[39] Xiao Qu, Myra B. Cohen, and Gregg Rothermel. 2008. Configuration-aware Regression Testing: An Empirical Study of Sampling and Prioritization. In *International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 75–86.

[40] Elnatan Reisner, Charles Song, Kin-Keung Ma, Jeffrey S. Foster, and Adam Porter. 2010. Using symbolic evaluation to understand behavior in configurable software systems. In *International Conference on Software Engineering*. IEEE, 445–454.

[41] Brian Robinson, Mithun Acharya, and Xiao Qu. 2012. Configuration Selection Using Code Change Impact Analysis for Regression Testing. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*. IEEE Computer Society, 129–138.

[42] Christian Schmitt, Sebastian Kuckuk, Frank Hannig, Jürgen Teich, Harald Köstler, Ulrich Rüde, and Christian Lengauer. 2016. Systems of Partial Differential Equations in ExaSlang. In *Software for Exascale Computing - SPPEXA (Lecture Notes in Computational Science and Engineering)*, Vol. 113. Springer, 47–67.

[43] Judith Segal and Chris Morris. 2008. Developing scientific software. *IEEE Software* 25, 4 (2008), 18–20.

[44] Norbert Siegmund, Alexander Grebhahn, Christian Kästner, and Sven Apel. 2015. Performance-Influence Models for Highly Configurable Systems. In *Proceedings of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM Press, 284–294.

[45] Charles Song, Adam Porter, and Jeffrey S. Foster. 2014. iTree: Efficiently Discovering High-Coverage Configurations Using Interaction Trees. *IEEE Transactions on Software Engineering* 40, 3 (March 2014), 251–265.

[46] Cemal Yilmaz, Myra B. Cohen, and Adam Porter. 2006. Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering* 31, 1 (2006), 20–34.

[47] Linbin Yu, Yu Lei, Raghu N. Kacker, and D. Richard Kuhn. 2013. ACTS: A Combinatorial Test Generation Tool. In *International Conference on Software Testing, Verification and Validation*. 370–375.