doepipeline: a systematic approach for optimizing multi-level and multi-step data processing workflows

Daniel Svensson[1,$], Rickard Sjögren[1,2,$], David Sundell[3], Andreas Sjödin[3], Johan Trygg[1,2,*]

1. Department of Chemistry, Computational Life Science Cluster (CLiC), Umeå University, Umeå, Sweden
2. Corporate Research, Sartorius AG, Umeå, Sweden
3. Division of CBRN Security and Defence, FOI - Swedish Defence Research Agency, Umeå, Sweden

$ These authors contributed equally to this work

* Corresponding author

Johan Trygg: johan.trygg@umu.se

Abstract

Background

Selecting proper parameter settings for bioinformatic software tools is challenging. Not only will each parameter have an individual effect on the outcome, but there are also potential interaction effects between parameters. Both of these effects may be difficult to predict. Making the situation even more complex, multiple tools may be run in a sequential pipeline where the final output depends on the parameter configuration of each tool in the pipeline. Because of the complexity and difficulty to predict outcome, parameters are in practice often left at default settings or set based on personal or peer experience obtained in a trial and error-fashion. To allow reliable and efficient selection of parameters for bioinformatic pipelines, a systematic approach is needed.

Results

We present *doepipeline*, a novel approach for optimizing bioinformatic software parameters, based on core concepts of the Design of Experiments methodology and recent advances in subset designs. Optimal parameter settings are first approximated in a screening phase using a subset design that efficiently span the entire search space, and subsequently optimized in the following phase using response surface designs and OLS modeling. Doepipeline was used to optimize parameters in three use cases; 1) de-novo assembly, 2) scaffolding of a fragmented assembly, and 3) k-mer taxonomic classification of nanopore reads. In all three cases, *doepipeline* found parameter settings producing a better outcome with respect to the measured characteristic when compared to using default values. Our approach is implemented and available in the Python package *doepipeline*.

Conclusions

Our proposed methodology provides a systematic and robust framework to optimize software parameter settings, in contrast to labor- and time-intensive manual parameter tweaking. The implementation in *doepipeline* makes our methodology accessible and user-friendly, and allows for automatic optimization of tools in a wide range of cases. The source code of *doepipeline* is available at https://github.com/clicumu/doepipeline and is installable through conda-forge.

Keywords

Design of Experiments, Optimization, Sequencing, Nanopore, MinION, Assembly, Classification, Scaffolding

Background

Bioinformatic software tools frequently expose a number of outcome-related parameters for the user to set or change from their default values. These parameters may be different forms of input filters, or alter the behavior of the running algorithm. Parameters may be either quantitative or qualitative (multi-level) in nature. While it is beneficial to customize tools to the specific situation, it is not always obvious what effect changing parameters will have on the outcome. This may be due to lacking documentation, poor understanding of the algorithm, or interaction effects between parameters that are difficult to oversee. Additionally, it is common that software tools are combined into pipelines, for example when calling genetic variants from raw sequence reads [1,2]. Pipelining tools in this manner further increase the complexity of selecting optimal parameter settings by both increasing the number of parameters and potential interaction effects. The settings for a particular data processing pipeline may also have to be tailored to the type of technology that was used to generate the data, for example the different platforms available for sequencing DNA that yield different error profiles [3]. In all, the strategy for selecting parameter settings therefore typically consists of using values derived from personal or peer experience obtained in a trial-and-error fashion, or simply using the default values. This kind of non-systematic parameter setting selection run the risk of producing sub-optimal results.

The combined ranges of all possible parameter settings form a parameter space. To find the optimal point in the parameter, exhaustive brute-force search by simply trying all possible combinations is guaranteed to find the optimum. Since the number of combinations increase exponentially, exhaustive search quickly becomes infeasible as the number of parameters, and their ranges, grows. Instead, statistical Design of Experiments (DoE) can be used to span and investigate the parameter space in an efficient manner [4]. DoE aims to maximize the information gain while minimizing the number of required experiments [5]. This is done by

introducing variation to the investigated system in a structured manner in order to explain how the parameters (*factors*) influence the result (*response*). The variation is introduced according to statistical designs to simultaneously vary the factor settings at a specific set of values (*levels*) and modeling the system using statistical methods, for example with ordinary least squares regression (OLS) [5,6]. The simplest type of statistical design is the full factorial design (FFD) where all combinations of factor levels are investigated in an exhaustive manner, meaning that they quickly become infeasible. To greatly reduce the number of required experiments, fractional factorial designs (FrFD) investigate a structured subset of the FFD [6]. The problem is that FrFD are not trivial to use in situations where there are more than 2 levels to investigate, and that there is no obvious way to combine qualitative and quantitative variables. Recently, fractional factorial designs were generalized into the so called generalized subset design (GSD) [7]. GSDs are balanced and near-orthogonal multi-level and multi-factor subset designs capable of mixing quantitative and qualitative factors, allowing for investigating a large and diverse set of parameters in an efficient manner.

Although DoE is primarily used within analytical chemistry, a DoE approach has previously been used by Eliasson et al to optimize software parameter settings in a LC-MS metabolomics data processing pipeline [8]. In essence, this approach consists of sequentially updating a statistical design based on the predicted optimal configuration of settings, until converging at an optimum. We build upon the approach proposed in Eliasson et al, and have developed a strategy for automated optimization of software parameter settings. We extend Eliasson et al's approach with a screening phase using the recently developed GSD to efficiently span a much larger parameter space. We also make it possible to optimize multiple responses simultaneously. Our strategy is software-agnostic and is implemented as a user-friendly Python package - *doepipeline*.

In this article, we outline our DoE-based strategy for a systematic approach for optimizing multi-level and multi-step data processing workflows, and exemplify the application of *doepipeline* in three cases; 1) de-novo assembly of a bacterial genome, 2) scaffolding of contiguous sequences (contigs) of a bacterial genome using 3rd generation sequencing (nanopore) data, and 3) k-mer classification of long noisy sequence reads generated by ONT MinION sequencing units.

Methods

We propose an approach for optimization of software parameters, based on methods of statistical design of experiments. Our approach has been implemented in a python package (*doepipeline*), and can be divided into two distinct phases:

1. Screening using a generalized subset design to find an approximate optimum and best choice of categorical variables.
2. Iterative optimization starting from the best point found by screening based on the algorithm by Eliasson et al [8].

Screening and optimization phases are schematically described in Figure 1 and described in more detail in the following subsections. Prior to screening and optimization, the user specifies what parameters to use as factors in the designs, if they are categorical or numerical, and allowed categories or value spans that is investigated. The user also specify what process outcomes to use as response, if it should be maximized, minimized or reach a target value. In cases where several responses are used the user also needs to specify low/high limits and target for each response. The responses are then re-scaled according to the limits and targets and combined into a single response using the geometric mean according to Derringer & Suich desirability functions [9].

Figure 1 - Schematic visualization of *doepipeline* design space movement. Example of an optimization of two factors (A and B) through both the screening and the optimization phase, completed in 3 iterations. All dots represent an executed pipeline with the parameters set by factors A and B. Triangles represent executed pipelines using the optima of an Ordinary Least Squares (OLS) model calculated in each optimization iteration. Red dots and triangles represent the best found configuration of factors in each iteration. Dashed lines represent the current high and low parameter settings in each iteration. *Screening phase*: a GSD using three levels and a reduction factor of 2 is used to span the design space. The pipelines are executed with the factor configurations suggested by the GSD and an approximate optimum is found (red dot). *Optimization phase*: in iteration 2, an optimization design is created around the best found configuration in the screening phase (black dots). In iteration 3, the design space is moved in the direction of the configuration of factors that produced the best result (red triangle) in iteration 2. *Doepipeline* halts when the best response is produced by a configuration of factors that lies close to the center point (red triangle in iteration 3).

Screening for Approximate Optimum

The purpose of the screening phase is to span the full search space to find regions with close to optimal performance. Screening is performed by executing the specified pipeline using combinations of factor configurations given by a GSD. Using a GSD effectively reduces the number of experiments to run, while optimally spanning the search space (Fig. 1a). The number of experiments required to investigate a given set of factors at a number of levels is approximately an integer fraction of the total number of possible combinations, which depends on the number of factors and their levels. A higher number of levels increase the resolution of the searched space during screening but also exponentially increase the number of required runs. We have found that five levels per numeric factor span large search spaces

with a high enough resolution to give satisfactory results, but it is possible to set the number of levels individually for each factor in *doepipeline*. Similarly, we have found that the integer fraction of the full design that the GSD should represent can be safely set at the number of factors included in the design. However, this may also be controlled by the user by setting the *reduction factor* setting in *doepipeline*.

The screening phase also serves the purpose of setting the category to use for categorical variables. For later optimization, qualitative factors are fixed at the category of the best factor configuration according to the screening. By fixing qualitative factors, only numeric factors are investigated during the following optimization phase.

Optimization of Numeric Factors

After selecting the best factor configuration during screening, numerical factors are optimized using response surface designs. A response surface design, for instance a central composite design, is constructed around the best found configuration. That is, the configuration found to produce the best result during the screening phase is used as center point in the new response surface design (Fig. 1b). If the configuration lies at the edge of a factor's global design space, the level closest to it in the GSD is chosen as center point instead. For the new design, high and low settings of each numeric factor are set to be the midpoint between the closest screening levels above respectively below the chosen center point, as indicated by the dashed line in Figure 1b. The span of each factor is then defined as the difference between high and low settings. Similar to during screening, the specified pipeline is executed using factor configurations given by the response surface design.

During each optimization iteration the pipeline performance is approximated using Ordinary Least Squares (OLS) regression [10]. By fitting a regression model the optimal configuration can be found by optimizing the predicted response according to the model. The factors

included in the OLS model are selected either using a best subset approach or by using greedy forward selection, the latter is preferred when more than four factors are included in the design. If the predictive power (Q2) of the model is acceptable (Q2 > 0.5), the model is used to predict an optimal parameter configuration. Each numeric factor's settings are then updated based on the best result in a manner similar to the algorithm given by Eliasson et al [8]. For each factor, the difference between the predicted best factor setting and the factor center point is calculated. If this distance is larger than 25% of the span of the factor, the high and low settings of the factor are updated in the direction of the best result. The default step length is 25% of the span of the factor, i.e. the high and low settings are moved 25% of the step length (Fig. 1b, iteration 3). We found the algorithm did not always converge at this stage, and moved the design space back and forth between iterations. To alleviate this problem, we implemented design space *shrinkage* that shrinks the design space span between iterations, and found that it successfully improved convergence. If the updated factor settings are proposed to lie outside the predefined design space limits, the design is placed on the factor limits keeping the factor span. If the design has not moved between two iterations, or the best response is not improved upon compared to the previous iteration the algorithm has converged and halts. If the optimization algorithm halts and responses have not reached their minimally acceptable values, the screening results are re-evaluated and a new optimization phase is run using the next best screening results. At the end of the optimization iterations the factor configurations that has produced the best result throughout the iterations is chosen as the optimal configuration.

Generation of sequence data

In total 24 sequence libraries for nanopore sequencing were created for selected DNA extractions using the SQK□LSK108 Ligation Sequencing Kit according to manufacturer's

specifications. 5-6 of the 24 samples were barcoded and sequenced together using a FLO□MIN107 flow cell. In total 4 MinION cells were used for the sequencing of the samples. The quality control indicated degradation for 2 samples so they were excluded from the MinION sequencing. The same 24 samples were also sequenced using a 600 bp sequencing kit with Nextera XT library preparations on an Illumina MiSeq machine. The *Francisella tularensis sp. holarctica* sample FSC200 (NCBI genome assembly ASM16877v2) [11] was chosen as an example dataset in case 1 and 2 of this study. In addition, the FSC200 sample was sequenced using an Illumina HiSeq 2500 machine. This 2x100 bp dataset were subsampled down to an estimated coverage of 100X (1.9M 100 bp reads), from the original coverage of approximately 658X. Subsampling was performed with *seqtk* [12] (v. 1.2-r94, installed through bioconda [13]).

Case 1: de-novo assembly of a bacterial genome

In this example we optimize the software ABySS [14,15] (v. 2.0.2, installed through bioconda [13]), a paired-end sequence assembler, to assemble the genome of an isolate of *Francisella tularensis* ssp. *holarctica* (FSC200). ABySS has in total 27 different parameters that can be specified by the user. Some are directly related to the running time and memory usage of the software (such as number of threads to use, or the bloom filter size), while others are related to the quality and/or characteristics of the resulting assembly (such as the size of k-mer or the minimum mean k-mer coverage of a unitig). For this example we have focused on the latter type of parameter. Hence, all parameters chosen to be part of the optimization were deemed to potentially have an effect on the resulting assembly. The chosen parameters were: size of k-mer (*k*) (KMER), minimum mean k-mer coverage of a unitig (*c*) (MIKC), minimum alignment length of a read (*l*) (MIAL), and minimum number of pairs required for building contigs (*n*) (MIPA).

For this optimization we set the investigated factor space so that the default value for each factor was included within the span of each factors' min and max values (Table 1). Although central to the ABySS algorithm, there is no default value for the k-mer size parameter. But since the value of the k-mer size is bounded by the actual read length it was still possible to define the GSD search space in a satisfactory way. For comparison purposes we have however considered a k-mer size of 31 as the default setting. In this example we ran the initial screening with a reduction factor of 8, and used Central Composite Face-centered (CCF) designs in the following optimization iterations. We used a shrinkage factor of 0.9 (*-s*), and set the model selection method (*-m*) to greedy to speed up the model selection. All other *doepipeline* settings were kept at default values.

There are many metrics that can be used to evaluate the quality of a de-novo assembly, and which specific ones to use depends on what the assembly is to be used for [16,17]. Example metrics include the number of resulting contiguous sequences (contigs), the total sequence covered by the assembly, and the N50 value. The N50 value is the length of the contig that, when ordering the contigs by size, spans the midpoint of the total assembly. Hence, the N50 value can be viewed as an assessment of the quality of the assembly in terms of contiguity.

We used the total size of the assembly (tSeq), the number of contigs (nSeq), and the N50 value as responses. Since this optimization contained multiple responses, it is required to set low/high acceptable limits of each response, as well as target values to reach. The low and high limits for the responses were set with respect to the obtained result using the default settings on the same input data, meaning that worst acceptable results are the default results. The target for the tSeq response was set to the reference genome size for FSC200 [11], while the targets for the nSeq and N50 responses were set to values that was considered achievable (Table 2).

The input data to ABySS was the subsampled Illumina HiSeq 2500 sequence data of FSC200 (see Generation of sequence data). Prior to calculating the values for the responses we applied a length-based filter on the assembly using *Fastaq* [18] (v. 3.17.0), keeping only the contigs larger than 1000 bp in length. This filter was also applied when calculating the response from the pipeline using the default parameter configuration. The software *seqstats* [19] was used to calculate the response values from the filtered assembly.

Case 2: scaffolding of a bacterial genome assembly using long reads

Assembling a genome with short reads typically results in a fragmented assembly, consisting of a number of contigs. However, the way these contigs are connected with each other - both in terms of ordering, distance, and direction - remains unknown. The cause of the fragmentation is that certain stretches of genomes have a low complexity and are therefore impossible to resolve with short reads. One way of stitching together the contigs of an assembly is by using paired reads with long insert sizes, or as is increasingly common - using long reads from for example Nanopore or PacBio platforms. The long reads have an increased chance of spanning the low-complexity regions, effectively anchoring both ends of a pair of contigs together and thus resolving the gap. The process of connecting contigs together is referred to as scaffolding, and the resulting sequences are known as scaffolds.

SSPACE-LongRead [20] (SSPACE) use long reads, such as those produced by PacBio or Nanopore platforms, to scaffold an assembly. When running the software the user can manipulate a total of six parameters that relate to the resulting scaffolds. We investigate if manipulating some of the parameters would yield a better result than by running SSPACE (v. 1-1) with default parameter settings. We chose to optimize the minimum alignment length to allow a contig to be included for scaffolding (*a*) (ALEN), the minimum gap between two contigs (*g*) (GLEN), the maximum link ratio between two best contig pairs (*r*) (RRAT), and

the minimum identity of the alignment of the long reads to the contig sequences (*i*) (IDEN). As response, we maximized the N50 value of the resulting scaffolded assembly.

We set the investigated space for the factors so that the default value for each factor was included within the span of each factors min and max values (Table 3). For the optimization phase following the screening phase we chose to use a CCF design for the experiments. The reduction factor for the GSD was kept at default value, i.e. the number of factors in the investigation, which in this case was 4. The model selection method (-*m*) was set to greedy and the shrinkage factor (-*s*) to 0.9. All other *doepipeline* settings were kept at default values.

The input assembly had been constructed with ABySS [15] (v. 2.0.2) (k=71) and subject to a contig length filter (>1000 bp). It consisted of 94 contigs between 1,685 and 87,479 bp, had a N50 of 27,549 bp, and totaled 1,800,912 bp prior to scaffolding. The assembly was constructed using the FSC200 Illumina HiSeq 2500 sequence data (see Generation of sequence data). The read set used for scaffolding consisted of 132,258 nanopore reads between 163 and 108,214 bp (N50=679 bp), totaling 104,374,862 bp. Seqstats [19] was used to calculate the response from the scaffolded assembly.

Case 3: k-mer classification

K-mer classification is a method to assign taxonomic labels to short DNA reads [21]. The method requires a precomputed database of k-mers generated from previously known and assembled genomes, for example all complete genomes in the NCBI database. When classifying a sample, the k-mer set of each read is calculated and compared with the database of known k-mers. The read is then assigned to the most specific taxonomic class within the database using the highest scoring k-mer root-to-leaf classification path following the taxonomic hierarchy. This method is for example implemented in the software Kraken [22].

Kraken also uses a least common ancestor method, which reclassify reads that are assigned to multiple taxonomic sub-classes under a parent node. A read with non-unique leaf assignment will then be assigned to the least common ancestor where there is little or no assignment conflicts instead. The k-mer classification method implemented in Kraken can be applied on longer error prone reads even if it is optimized for short accurate reads. However, it will be less accurate due to the different error frequencies and therefore exhibit increased false positive rates.

In this study we used the software KrakenUniq [23] (v. 0.4.8), which at the time was called KrakenHLL. KrakenUniq builds upon the Kraken engine but additionally records the number of unique k-mers as well as coverage for each taxon. Three factors were used in the optimization: k-mer size (KSIZE), precision (PRES) and a filter (FILT). It should be noted that PRES was deprecated in later versions above 0.4.8. We included the default settings in the investigated design space (table 4) and used a CCF design for the optimization phase. The reduction factor for the GSD was kept at its default value, in this case 4. The model selection method (-*m*) was set to greedy and the shrinkage factor (-*s*) to 0.9. All other *doepipeline* settings were kept at default values. The response was the F1 score (Eq. 1), which is the harmonic mean of precision and recall.

$$F1 = 2 \cdot \frac{(precision \cdot recall)}{(precision + recall)} \qquad \textbf{(Eq. 1)}$$

The input data was nanopore sequenced reads from several *Yersinia pestis* and several near neighbour *Yersinia* (*entereolitica*, *kristensenii*, *similis* etc.). The dataset consisted of 231,578 *Y. pestis* reads and 500,443 non- *Y. pestis* Nanopore sequencing reads (see Generation of sequence data).

Doepipeline

Implementation

Doepipeline is fully implemented in the Python programming language and source code is available for download at github (https://github.com/clicumu/doepipeline) and installable with conda-forge [24,25] and through PyPi. Generation of statistical designs are made through the python package PyDOE2 [26], in which the GSD has been implemented.

Usage

Configuration of the optimization is done in a structured YAML file with sections for the experimental design and for the pipeline steps (commands) to run. The design section include the names of the investigated factors and their min/max values (design space), the responses and their goals (minimize/maximize), and the type of design to use in the optimization phase. The pipeline section is where each individual pipeline step is specified. In each iteration, *doepipeline* takes the pipeline steps as configured and substitutes the parameters under investigation with the values given by the statistical design. A batch script is created for each pipeline step, with any parameter values substituted, and the execution of it is controlled by *doepipeline*. Pipeline steps are executed either in parallel mode, where all experiments are run at the same time, or in sequential mode where each pipeline with all of the steps are executed in sequence. For reference, we provide example YAML files at the github repository.

Scientific data processing today can include vast amounts of data and/or require substantial computing power. In those cases data processing is commonly performed on compute clusters that typically use some queueing system in order to handle all user requests for resources. An example of such a queueing system is the Slurm Workload Manager [27] (Slurm). To accommodate users of compute clusters, we have implemented Slurm support for *doepipeline*. If using Slurm, specify the Slurm options in the YAML file as you would when

running a regular Slurm job. The Slurm options are transferred by *doepipeline* to the batch script which is then submitted to Slurm using *sbatch*.

After optimization, the parameter values suggested by *doepipeline* are saved in the working directory of the optimization. Additionally, there is a rich log file that can be investigated to follow the workflow.

Results

Case 1: de-novo assembly of a bacterial genome

The goal of de-novo assembly is to combine raw sequence reads into a representation of an organism's genome, i.e. to obtain an as contiguous genomic sequence as possible. Due to characteristics of the genome sequence itself, in combination with short reads, this process can be difficult. For example, sequence reads from less complex segments of the genome will map to more than one position, causing ambiguities that are not possible to resolve which in turn leads to fragmentation of the assembly.

A popular assembly software is ABySS [15], which exposes 27 different user-controlled parameters. We set up an example for optimization of de-novo assembly software parameters using ABySS (see methods section). *Doepipeline* ran for two iterations before halting. Thus, the best response was obtained in the first iteration, in the GSD screening phase. The experimental sheet and corresponding response values from the GSD screening and iteration 2 are included as Additional file 1 and 2, respectively. Using the optimized parameter settings (Table 1) we obtained a 1.6% and 13.1% increase of the investigated responses tSeq and N50, and a 2.2% reduction of nSeq as compared to when running abyss-pe with default settings (Table 2).

Table 1 - Factors in the de-novo assembly case. The four factors investigated in the de-novo assembly case are described. The letter in parenthesis following the parameter name is the parameter used in the abyss-pe command line interface. Min and max values define the design space. *: There is no default value explicitly specified by the ABySS documentation. However here we have used a k-mer size of 31 for comparison purposes. **: This refers to the square root of the median k-mer coverage, which is affected by the sequencing depth and choice of k-mer size.

Parameter	**Abbr.**	**Type**	**Min**	**Max**	**Default**	**Optimized**
Size of k-mer (*k*)	KMER	Ordinal	20	90	31*	38
Minimum mean k-mer coverage of a unitig (*c*)	MIKC	Quantitative	2	15	sqrt(median)**	8.5
Minimum alignment length of a read (*l*)	MIAL	Ordinal	20	60	40	30
Minimum number of pairs required for building contigs (*n*)	MIPA	Ordinal	5	15	10	15

Table 2 - Responses in the de-novo assembly case. The three responses that were measured in the de-novo assembly case are described. *: Responses that has the criterion maximize has a low limit, and those with criterion minimize has a high limit. **: Default values are based on using k-mer size of 31 and leaving all other parameters as is.

Response	**Abbr.**	**Criterion**	**Low/high limit***	**Target**	**Default****	**Optimized**
Total sequence in assembly (bp)	tSeq	Maximize	1,830,000	1,894,157	1,835,427	1,864,165
Number of contigs in assembly	nSeq	Minimize	95	85	91	89
N50	N50	Maximize	28,000	35,000	28,149	31,847

Case 2: scaffolding of a bacterial genome assembly using long reads

Scaffolding is the process of connecting contigs obtained from an assembly step with each other. In this example we aimed at optimizing parameters for the scaffolding software SSPACE-LongRead [20], which relies on long reads to span the low-complexity regions that are typically found between the contigs of an assembly. *Doepipeline* ran for 3 iterations before halting, obtaining the best result in iteration 2. The response values and investigated parameter settings in each iteration are included in Additional files 3-5, respectively. The response (N50) value obtained when using the default parameter settings was 1,141,889 bp. Using the optimized parameter settings (Table 3) resulted in a 66.9% increase in the response (1,905,883 bp).

Table 3 - Factors in the scaffolding case. The four factors investigated in the scaffolding case are described. The letter in parenthesis following the parameter name is the parameter used in the SSPACE command line interface. Min and max values define the design space.

Parameter	**Abbr.**	**Type**	**Min**	**Max**	**Default**	**Optimized**
Minimum alignment length to allow a contig to be included for scaffolding (*a*)	ALEN	Ordinal	0	5000	0	0
Minimum gap between two contigs (*g*)	GLEN	Ordinal	-3000	3000	-200	-750
Maximum link ratio between two best contig pairs (*r*)	RRAT	Quantitative	0.1	0.7	0.3	0.325
Minimum identity of the alignment of the long reads to the contig sequences (*i*)	IDEN	Ordinal	30	90	70	82

Case 3: k-mer classification

K-mer classification is used to gather information about the general species content of a metagenomic sample. It is possible to visualize the general distribution of species through classified reads or to identify presence/absence of reads classified to specific targets. By using third generation sequencing techniques, such as Oxford Nanopore, it is possible to classify reads from an unknown sample in real time. But due to the long error prone reads from the third generation sequencing machines, there is a greater risk of misclassification. On Genus level this is usually no problem. But when it comes to discriminating pathogenic from non-pathogenic species misclassification may become problematic, in particular false positive signals of pathogenic species. We investigated the KrakenUniq [23] (v0.4.8) algorithm and used *doepipeline* to find optimized settings for long error prone reads to increase the fraction of true positives to false positives using the F1-score as response. KrakenUniq also has a filter that may reduce the amount of false positive reads. The filter will adjust each assigned read up the tree until the desired threshold is met, where the threshold is the number of assigned k-mers over the number of unique k-mers in that category [23].

Doepipeline results shows that a selection of parameters that differs extensively from default (Table 4) reduces the risk of read misclassification to non-pathogenic species while still maintaining an accurate classification of the target pathogen.

Optimization ran for two iterations before halting and the best results were found during the first iteration (GSD screening phase). The experimental sheet and corresponding response values from the GSD screening and iteration 2 are included as Additional file 6 and 7, respectively. Using the optimized parameter settings we could increase the F1 score from 0.82 to 0.99, compared to when running KrakenUniq with default settings.

Table 4 - Factors in the k-mer classification case. The three factors investigated in the k-mer case are described. Min and max values define the design space.

Parameter	**Abbr.**	**Type**	**Min**	**Max**	**Default**	**Optimized**
K-mer size of the database	KSIZE	Ordinal	25	31	31	25
Standard deviation on the relative errors of the estimate	PRES	Ordinal	10	18	12	18
Minimum taxID score threshold	FILT	Quantitative	0	0.05	0	0.05

Discussion

Selecting parameter settings for a data processing pipeline is complex, since the influence of the parameters on the end result is not always obvious. While personal and peer experience should not be underestimated, our approach provides a systematic way of determining the optimal settings. Specialized tools to optimize particular bioinformatic softwares have been proposed previously. For example, VelvetOptimizer [28] can be used to optimize the k-mer and coverage cutoff parameters of the Velvet assembler [29] and KmerGenie can be used to make an informed decision on the choice of k-mer in de Bruijn based assemblers [30]. A generalized, software-agnostic optimization approach is however preferred. Especially when several tools are used together in a pipeline.

Here we have presented such a generalized strategy for automated sequential optimization of software parameters, employing core concepts of DoE methodology. We have furthermore implemented our strategy in a user-friendly python package, *doepipeline*. The optimization strategy and the use of *doepipeline* was exemplified in three bioinformatics use cases; de-novo assembly of a bacterial genome using Illumina reads, scaffolding a bacterial genome

assembly using nanopore reads, and k-mer classification of metagenomic third generation sequencing data. In all three cases, we saw an improvement in the measured response variables as compared to when using the default parameter settings. The improvement of the measured responses in our examples ranged between 1.6 and 66.9%.

One of the advantages of our proposed strategy is the use of a GSD in a screening phase prior to the optimization phase. Compared to Eliasson et al [8], we are able to screen a much larger design space efficiently prior to optimization using the GSD-based approach. In order for the optimization phase to converge in a feasible number of iterations, the design space should be restricted somehow. Deciding the range of the factors without guidance risks creating a too narrow or wide design space. Instead, the screening allows the user to set up a relaxed (wide) design space to investigate and to approximate the optimal factor combination. The approximation represents a substantiated initial center point to set up a more narrow optimization design around. The screening phase will also identify promising values of any qualitative factors and fix them before optimization. Thus, the GSD screening phase can be viewed as a systematic approach for restricting the design space for the subsequent optimization phase.

The fraction of the full design that a GSD represents can be controlled in *doepipeline* with the reduction factor parameter. We ran the optimization of ABySS with a GSD reduction factor of 8, but another optimization of ABySS where a reduction factor of 10 was used produced the same response values (data not shown) in fewer experimental runs (45 as opposed to 70). In addition, there is a degree of overlap among the response values in the GSD iterations (Additional files 1, 3, and 6). We also saw that the approximated optima was in fact not improved upon in the following optimization phase for two out of three of our examples (de-novo assembly and k-mer classification). In all, this could indicate that it is meaningful to try

running the GSD with a higher reduction factor than the recommended default, and/or reducing the number of levels, further reducing the number of experiments.

During development and testing of *doepipeline* we saw the design space moving back and forth between iterations in the optimization phase. We hypothesized that this behavior was due to that the underlying function was not modeled properly or that the function was flat within the investigated design space. To counteract this behavior we implemented three features; i) no prediction of the optimal factor combination if the predictive power (Q2) of the model was low (default: Q2<0.5), ii) validate the predicted optimal factor combination by carrying out the pipeline with those factor settings, and iii) shrink the span of the factors between iterations. After implementing these three features, *doepipeline* consistently converge to satisfactory results.

Specifying the pipeline in a YAML file allows for flexible configurations of commands to be run, essentially enabling optimization of any pipeline run on the command line. However, the number of parameters will typically increase with the length of the pipeline under investigation. At the same time there is a soft constraint on the number of parameters that can be investigated simultaneously. This constraint will be related to the problem currently under investigation and depends on the computational complexity of the pipeline, and the available computational and time resources. Instead of doing a global optimization of parameters, i.e. optimizing the entire pipeline at once, an alternative approach is to run sequential optimizations where you optimize only a piece of the pipeline at a time while keeping the default parameter values for the rest of the pipeline [8]. This type of sequential optimization is not fully implemented in *doepipeline* and remains a feature for future updates. Sequential optimization of a pipeline currently requires that an optimization is set up for each step of the pipeline and that the optimized parameter values are manually updated for following steps of the pipeline.

Conclusion

Our proposed strategy represents a systematic approach for optimization of software parameters. Our implementation in the software-agnostic and user-friendly *doepipeline* could potentially serve as a starting point for experimenters and bioinformaticians who today rely on default settings or common practice for their data processing pipelines.

Declarations

Ethics approval and consent to participate

Not applicable

Availability of data and material

The dataset generated and analyzed during the current study are available in NCBI BioProject accession PRJNA510899.

Consent for publication

Not applicable

Competing interests

The authors declare that they have no competing interests.

Funding

This work was funded by the Knut and Alice Wallenberg Foundation (DSv, JT) (2011.0042), the Swedish Research Council (DSv, RS, JT) (2016□04376), Sartorius AG (RS, JT), the Swedish National Strategic e□Science Research Program eSSENCE (RS, JT), the Swedish Ministry of Defence (DSu, AS) (FOI project A404018) and Swedish Civil Contingencies Agency (DSu, AS) (FOI project B4662 B2Forensics).

Authors' contributions

DSv, AS and JT initiated the study, DSv and RS designed the algorithm and implemented the python code. DSv and DSu analyzed and interpreted the three example cases. All authors wrote, read and approved the final manuscript.

Acknowledgements

This study makes use of sequence data generated at the Swedish Defence Research Agency by Edvin Karlsson and Emelie Samuelsson-Näslund. The authors would like to acknowledge support from Science for Life Laboratory (SciLifeLab) and the National Genomics Infrastructure (NGI) for providing assistance in massive parallel sequencing. Computations were carried out at the Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX) and the High Performance Computing Center North (HPC2N), part of the Swedish National Infrastructure for Computing (SNIC). The authors thank Matilda Rentoft and Mattias Eliasson for fruitful discussions.

References

1. DePristo MA, Banks E, Poplin R, Garimella K V, Maguire JR, Hartl C, et al. A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nat Genet. NIH Public Access; 2011 May;43(5):491–8.
2. Van der Auwera GA, Carneiro MO, Hartl C, Poplin R, Del Angel G, Levy-Moonshine A, et al. From FastQ data to high confidence variant calls: the Genome Analysis Toolkit best practices pipeline. Curr Protoc Bioinforma. NIH Public Access; 2013;43(1110):11.10.1-33.
3. Li H. Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics. 2018;34(18):3094–100.
4. Fisher RA. The design of experiments. Edinburgh/London: Oliver and Boyd; 1935.
5. Eriksson L, Johansson E, Kettaneh-Wold N, Wikström C, Wold S. Design of experiments□:

principles and applications. Umeå: Umetrics Academy; 2008.

6. Box GEP, Hunter WG, Hunter JS. Statistics for experimenters□: an introduction to design, data analysis, and model building. New York: Wiley; 1978.

7. Surowiec I, Vikström L, Hector G, Johansson E, Vikström C, Trygg J. Generalized Subset Designs in Analytical Chemistry. Anal Chem. 2017;89(12):6491–7.

8. Eliasson M, Rännar S, Madsen R, Donten MA, Marsden-Edwards E, Moritz T, et al. Strategy for Optimizing LC-MS Data Processing in Metabolomics: A Design of Experiments Approach. Anal Chem. 2012 Aug 7;84(15):6869–76.

9. Derringer G, Suich R. Simultaneous Optimization of Several Response Variables. J Qual Technol. Taylor & Francis; 1980 Oct 22;12(4):214–9.

10. Dismuke C, Lindrooth R. Ordinary least squares. Methods Des Outcomes Res. ASHP Bethesda; 2006;93:93–104.

11. Svensson K, Sjödin A, Byström M, Granberg M, Brittnacher MJ, Rohmer L, et al. Genome sequence of Francisella tularensis subspecies holarctica strain FSC200, isolated from a child with tularemia. J Bacteriol. American Society for Microbiology (ASM); 2012 Dec;194(24):6965–6.

12. seqkt [Internet]. Available from: https://github.com/lh3/seqtk

13. Grüning B, Dale R, Sjödin A, Chapman BA, Rowe J, Tomkins-Tinch CH, et al. Bioconda: sustainable and comprehensive software distribution for the life sciences. Nat Methods. Nature Publishing Group; 2018 Jul 2;15(7):475–6.

14. Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ, Birol I. ABySS: A parallel assembler for short read sequence data.

15. Jackman SD, Vandervalk BP, Mohamadi H, Chu J, Yeo S, Hammond SA, et al. ABySS 2.0: resource-efficient assembly of large genomes using a Bloom filter. Genome Res. Cold Spring Harbor Laboratory Press; 2017;27(5):768–77.

16. Earl D, Bradnam K, St John J, Darling A, Lin D, Fass J, et al. Assemblathon 1: a competitive assessment of de novo short read assembly methods. Genome Res. Cold Spring Harbor Laboratory Press; 2011 Dec 1;21(12):2224–41.

17. Bradnam KR, Fass JN, Alexandrov A, Baranay P, Bechner M, Birol İİ, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. Gigascience. Oxford University Press; 2013;2(1):10.
18. Fastaq [Internet]. Available from: https://github.com/sanger-pathogens/Fastaq
19. seqstats [Internet]. Available from: https://github.com/clwgg/seqstats
20. Boetzer M, Pirovano W. SSPACE-LongRead: scaffolding bacterial draft genomes using long read sequence information. BMC Bioinformatics. BioMed Central; 2014 Jun 20;15(1):211.
21. Breitwieser FP, Lu J, Salzberg SL. A review of methods and databases for metagenomic classification and assembly. Brief Bioinform. 2017 Sep 23;
22. Wood DE, Salzberg SL. Kraken: ultrafast metagenomic sequence classification using exact alignments. Genome Biol. BioMed Central; 2014 Mar 3;15(3):R46.
23. Breitwieser FP, Baker DN, Salzberg SL. KrakenUniq: confident and fast metagenomics classification using unique k-mer counts. Genome Biol. BioMed Central; 2018 Dec 16;19(1):198.
24. conda-forge [Internet]. Available from: https://conda-forge.org/
25. doepipeline (conda-forge) [Internet]. Available from: https://anaconda.org/conda-forge/doepipeline
26. PyDOE2 [Internet]. Available from: https://github.com/clicumu/pyDOE2
27. Yoo AB, Jette MA, Grondona M. SLURM: Simple Linux Utility for Resource Management. Springer, Berlin, Heidelberg; 2003. p. 44–60.
28. VelvetOptimizer [Internet]. Available from: https://github.com/tseemann/VelvetOptimiser
29. Zerbino DR, Birney E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Res. Cold Spring Harbor Laboratory Press; 2008 May 1;18(5):821–9.
30. Chikhi R, Medvedev P. Informed and automated k-mer size selection for genome assembly. Bioinformatics. Oxford University Press; 2014 Jan 1;30(1):31–7.

Additional material

Additional file 1

Excel file (.xlsx).

Complete experimental sheet for case 1 iteration 1. Contains factor settings and response values for all experiments in this iteration.

Additional file 2

Excel file (.xlsx).

Complete experimental sheet for case 1 iteration 2. Contains factor settings and response values for all experiments in this iteration.

Additional file 3

Excel file (.xlsx).

Complete experimental sheet for case 2 iteration 1. Contains factor settings and response values for all experiments in this iteration.

Additional file 4

Excel file (.xlsx).

Complete experimental sheet for case 2 iteration 2. Contains factor settings and response values for all experiments in this iteration.

Additional file 5

Excel file (.xlsx).

Complete experimental sheet for case 2 iteration 3. Contains factor settings and response values for all experiments in this iteration.

Additional file 6

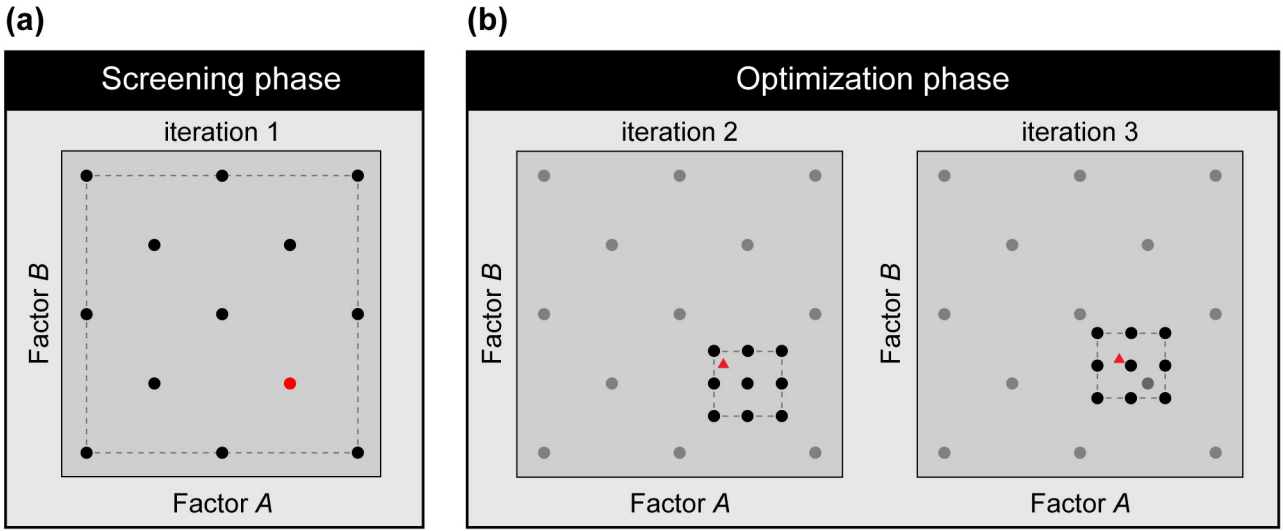
Excel file (.xlsx).

Complete experimental sheet for case 3 iteration 1. Contains factor settings and response values for all experiments in this iteration.

Additional file 7

Excel file (.xlsx).

Complete experimental sheet for case 3 iteration 2. Contains factor settings and response values for all experiments in this iteration.


(a)
Screening phase
iteration 1
Factor B
Factor A
(b)
Optimization phase
iteration 2
iteration 3
Factor B
Factor A
Factor B
Factor A