BOAssembler: A Bayesian Optimization Framework to Improve RNA-Seq Assembly Performance

Shunfu Mao(✉), Yihan Jiang, Edwin Basil Mathew, and Sreeram Kannan

University of Washington, Seattle, WA 98195, USA
{shunfu,yij021,edwin100,ksreeram}@uw.edu

Abstract. High throughput sequencing of RNA (RNA-Seq) can provide us with millions of short fragments of RNA transcripts from a sample. How to better recover the original RNA transcripts from those fragments (RNA-Seq assembly) is still a difficult task. For example, RNA-Seq assembly tools typically require hyper-parameter tuning to achieve good performance for particular datasets. This kind of tuning is usually unintuitive and time-consuming. Consequently, users often resort to default parameters, which do not guarantee consistent good performance for various datasets.

Results: Here we propose BOAssembler, a framework that enables end-to-end automatic tuning of RNA-Seq assemblers, based on Bayesian Optimization principles. Experiments show this data-driven approach is effective to improve the overall assembly performance. The approach would be helpful for downstream (e.g. gene, protein, cell) analysis, and more broadly, for future bioinformatics benchmark studies.

Availability: https://github.com/shunfumao/boassembler.

Keywords: RNA-Seq · Assembly · Bayesian Optimization

1 Introduction

Sequence assembly is a process to recover the original genomic sequences from their sampled reads. Based on sequence type (DNA/RNA) and the availability of reference genome, there are different assembly problems. In this study, we focus on reference-based RNA-Seq assembly, which is a critical step to understand gene, protein and cell functions.

Existing popular reference-based RNA-Seq assemblers include Cufflinks [3] and Stringtie [2]. They usually align reads onto reference genome first, and utilize the read alignments to build a graph where each node represents a genome region (exon) and each edge represents the connection between two nodes by some

S. Mao and Y. Jiang—Equal contributors.

C. Martín-Vide et al. (Eds.): AlCoB 2020, LNBI 12099, pp. 188–197, 2020.
https://doi.org/10.1007/978-3-030-42266-0_15

reads. They then traverse the graph to find paths as the reconstructed RNA transcripts.

These assembly problems are essentially NP-hard [7] and existing tools resort to heuristic methods. For example, from the graph, Stringtie will extract the heaviest paths iteratively. Due to the heuristic approaches, these methods usually require parameter tuning to achieve good performance for particular datasets. Since most users may not understand the meaning of the parameters well and tuning itself is tedious and time-consuming, they usually end up with default settings. An automatic tuning framework, therefore, is necessary.

In machine learning (ML), Bayesian Optimization (BO) is gaining a surge of interest as its usefulness in tuning hyper-parameters for modern deep learning systems [10,11]. BO is favorable for optimizing objective functions that are expensive to evaluate and are over continuous domains of less than 20 dimensions [12]. BO has been widely used in most deep learning systems such as Natural Language Processing (NLP) [13], Reinforcement Learning (RL) [14], and Channel Coding [15]. Depending on algorithms and programming languages, several popular BO packages have been developed, such as GPyOpt [16].

There are limited work to introduce BO into computational biology fields. Recently [17] applies BO to improve eQTL analysis. To the best of our knowledge, no work has introduced BO to assembly tasks yet, which are fundamentally graph problems with their own unique challenges. To fill this gap, we have developed BOAssembler, which is a framework able to incorporate existing assemblers (such as Stringtie) and BO methods (such as GPyOpt) to assist assembler developers and biologists to spend minimal efforts to obtain better assembly hyper-parameters automatically fine-tuned for particular datasets.

Our contributions include: (a) We firstly explore the BO methods in (reference-based RNA-Seq) assembly tasks. (b) Our designed experiments show that BO is overall effective to improve assembly. (c) An open source end-to-end framework (BOAssembler) is provided for the assembly community to use.

2 Methods

2.1 Assembly

There are two kinds of RNA-Seq assembly problems: de novo assembly and reference-based assembly. For de novo assembly, we only have RNA-Seq reads, which is common in non-model organisms. For reference-based assembly, there is additional knowledge on the genome of the organism. De novo assembly is apparently more challenging and typical tools (such as Trinity [4] and recently Shannon [5]) require much more computational resources and more complicated evaluations. As the first step to bridge assembly and BO, we focus on reference-based RNA-Seq assembler. In particular, we focus on the widely used Stringtie, as recommended in [6].

A typical reference-based RNA-Seq assembly includes aligning sampled RNA-Seq reads onto a reference genome using external tools such as STAR [1] etc. For Stringtie, a (splice) graph will be prepared where each node represents a

unique exonic region supported by aligned reads and edges indicate how nodes are bridged by reads. Graph traversal algorithms will be applied to find paths as transcripts to best explain the constraints from graph nodes and edges.

Since assembly problems are NP hard [7], existing algorithms take a lot of heuristics (predefined threshold values). Therefore, assembler performance heavily depends on its parameters. For example in Stringtie, the parameter '-f' sets a fractional threshold so that the predicted transcripts having a lower relative abundance level than this will be discarded; a reduced '-f' threshold therefore encourages transcripts to be retained to improve sensitivity.

Developers of assemblers typically tune parameters by intuition on a few datasets, and offer selected parameters for assembler users to use. As the assembly performance for various datasets are usually parameter dependent, a more systematic method of tuning parameter is needed. Naive approaches would include grid search or random search. However, because typical assembly parameters are continuous, and of around 10 to 20 dimensions, which could make grid search on all possible combinations prohibitive. Random search [9], on the other hand, are expensive to guarantee good coverage.

2.2 Bayesian Optimization

BO aims at maximizing a real-value black-box function $f(\theta)$ with respect to θ [8] in a gradient-free approach. BO consists of a statistical surrogate objective function to model the input-output relationship between θ and $f(\theta)$, and an acquisition function to decide what to sample next. Firstly BO evaluates randomly chosen K datapoints of θ, and fits the prior statistical objective model. Then BO iteratively updates the posterior model with newly acquired $f(\theta_k)$, and selects θ_{k+1} to evaluate according to posterior. BO is a systematic approach to explore the parameter space according to a Bayesian model with limited allowed evaluations (i.e. iterations).

Our BOAssembler primarily uses Gaussian Process (GP) with Matern Kernel [11] as a natural model for statistical objective function, and Expected Improvement (EI) as a commonly used acquisition function. Specifically, assume we want to sample a new datapoint θ, and our current best parameter is θ^*. Then the improvement is defined as $[f(\theta) - f(\theta^*)]^+$. Note that the improvement is positive only when $f(\theta)$ is larger than $f(\theta^*)$. Then the expected improvement can be taken under posterior distributions of f given $\theta_{1:k}$:

$$EI_k(\theta) = \mathbb{E}([f(\theta) - f(\theta^*)]^+ | \theta_{1:k}, f(\theta_{1:k})) \tag{1}$$

As the expected improvement can be computed in closed-form, we can select the point with largest expected improvement to sample: $\theta_{k+1} = \arg\max EI_k(\theta)$.

The procedure of iterative update, based on GP and EI, is described in Algorithm 1, which includes K datapoints for BO initialization, and T iterations to acquire.

Input: D, K, T
Output: Best parameter θ^*
Fit the GP with K initial samples $\theta_k, k \in \{1, ..., K\}$;
i=0;
while $i<T$ **do**
  Update the GP posterior probability distribution on f using all available data;
  Use EI to compute the θ_{i+1} with updated posterior distribution;
  Obtain $f(\theta_{i+1})$;
  $i = i + 1$;
end
Return θ^* with best performance;

Algorithm 1. Baysian Optimization Algorithm

2.3 Combine BO and Assembly

The Motivation to Combine. We can formulate the problem of assembly parameter tuning as follows. The reference-based assembler together with its performance evaluation can be represented as an abstract function $f(D, \theta)$, where D includes both the read alignments used for assembly and the reference transcriptome (a set of ground truth RNA transcripts) used for evaluation, and θ refers to the parameters of f. After read alignments are assembled with given parameter θ, the assembly output (a set of RNA transcripts) will be compared with the reference transcriptome, and the quality of assembly is measured by scalar metrics such as precision p and sensitivity s. $f(D, \theta)$ outputs an evaluation score based on p and s. Our goal is to find a global optimal θ which maximizes $f(D, \theta)$ under limited number of iterations, as running assembler per iteration is time consuming.

Bayesian optimization works well for black-box gradient-free global optimization with moderate dimensionality. It is thus favorable to apply BO to optimize assembler parameters due to the following reasons: (a) Empirically BO works well for parameters with moderate dimensionality (less than 20). This is consistent with assemblers which typically have this number of parameters. (b) $f(.)$ is continuous, and the parameter θ are correlated, and has well-defined feasible set. The assembler parameters usually have continuous values within certain ranges. (c) The function $f(.)$ is expensive to evaluate, thus to evaluate all possible combinations of parameters is prohibitive. Indeed, assembly tasks are time consuming. (d) $f(.)$ is a 'black-box', while gradient-based optimization methods cannot be applied. Assemblers typically do not have a gradient due to the usage of thresholds, which makes black-box method favorable.

The Architecture. Figure 1 illustrates the overall architecture of BOAssembler. There are two parts: the assembly part (e.g. $f(D, \theta)$) and BO part.

The assembly part wraps up the RNA-Seq reference-based assembler (here Stringtie), which takes fixed read alignments as well as adjustable assembler

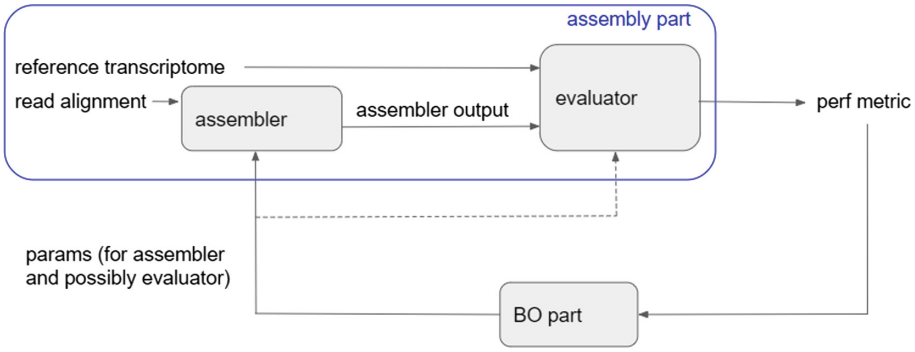

Fig. 1. BOAssembler architecture.

parameters as input, and outputs assembled RNA transcripts (in gtf format). In addition, the assembly part includes an evaluator block to access the assembly output. Basically, it calls the gffcompare[1] tool, which takes as input the assembly output and reference transcriptome, and outputs sensitivity and precision statistics. The sensitivity is the percentage of reference RNA transcripts that have been correctly recovered, and the precision means the percentage of assembled transcripts that correctly match the reference transcriptome. We further combine the sensitivity and precision (such as F1 score) as $f(D, \theta)$, to be used by the BO part. The evaluator may also take adjustable parameters as discussed in Sect. 2.3.

The BO part has its theory described in Sect. 2.2. The BO part mainly relies on GPyOpt, which implements the core BO methods (e.g. the GP + EI approach). The BO part treats the assembly part as a black box, where the input to the box is the parameters for assembler and evaluator, and the output of the box is the combined performance metric for the assembler (such as F1 score). The BO part will iteratively optimize the parameters for the black box function (e.g. the assembly part) based on the feedback of performance metric.

Metrics to Optimize. The assembly part outputs $f(D, \theta)$, which is a metric score and serves as an input to BO part. In particular, it is defined as a weighted F1 score ($S_w = \frac{\lambda p \times (1-\lambda)s}{\lambda p + (1-\lambda)s}$) on top of the evaluator's output in terms of sensitivity s and precision p of the assembly. $\lambda \in (0, 0.5)$ is also BO tunable.

There are several candidate metrics including the mean value ($S_m = \frac{s+p}{2}$) and the F1 score ($S_{F1} = \frac{2sp}{s+p}$). We found the BO part tends to overfit either s or p towards 1 when using S_m. Though S_{F1} is able to balance s and p, we find S_w is better to improve the final performance of sensitivity and precision. In our experiments, s tends to have a lower value range than p (due to many reference RNA transcripts do not have enough coverage), we hope to reward more for sensitivity improvement but still have gain on precision. Therefore, we

[1] https://ccb.jhu.edu/software/stringtie/gffcompare.shtml.

come up with the weighted F1 score S_w, which uses BO to figure out how much percentage we want to reward especially for the improvement of sensitivity.

Hyper-parameters. Our final goal is to find good assembler parameters. To achieve this, we hope to find hyper-parameters (θ) that achieves high metric score S_w. θ include both assembler parameters and evaluator parameter (λ). Each parameter has its name, type, default value and range. For example, Stringtie has a parameter '-f' with type float, default value 0.1 and range $(0.0, 1.0)$[2].

Usage and Extension of BOAssembler. To tune assembler parameters for a particular dataset in BOAssembler, the user only needs to provide a small sample of read alignments of the target dataset. The sampling can be done by our provided scripts. After some iterations, BOAssembler will report suggested parameters and its tuning history.

BOAssembler currently uses Stringtie as its default assembler. It supports Cufflinks as well. Extension to use other reference-based RNA-Seq assemblers is also straightforward. The user only needs to follow the Stringtie example, to add a line of Python code in a specified Python file, and to add a config file which contains the parameters to be tuned.

3 Result and Discussion

3.1 Datasets

Our goal is to use BOAssembler to tune assembler's hyper-parameters on a smaller dataset, and apply recommended hyper-parameters on a large assembly task. Since the smaller dataset has representative data of large assembly task, we expect tuned hyper-parameters can overall improve the large assembly task in terms of sensitivity and precision.

We build our results based on simulated datasets, since real datasets lack ground truth and it is hard to judge if an assembled RNA transcript is a false positive, or a new RNA transcript that has yet to be discovered. Consequently, the evaluator metric S_w is not feasible for real datasets. The simulated datasets are generated based on real ones in the following steps.

Firstly we prepare three real datasets, including: 132.05M Illumina single end reads (50-bp) sampled from human embryonic stem cells (HESC) (GSE51861, used in [18]), 115.36M Illumina pair end reads (101-bp) sampled from Lymphoblastoid cells (LC) (SRP036136, used in [19]), and 183.53M Illumina pair end reads (100-bp) sampled from HEK293T (Kidney) cells (SRX541227), previously produced and studied in StringTie [2].

Secondly, we use RSEM [20] to generate simulated reads from real datasets. To begin with, we choose LC reference transcripts (containing 207266 RNA

[2] See http://ccb.jhu.edu/software/stringtie/index.shtml?t=manual for an example of Stringtie's parameters.

transcripts) as the ground truth reference transcriptome annotations. We then do quantification of real datasets using RSEM and get learned statistics from real datasets. Based on learned statistics, we use RSEM to sample simulated reads from ground truth reference transcriptome. The simulated HESC has 150M 50-bp single-end reads, the simulated LC dataset has 150M 101-bp pair end reads and the simulated Kidney dataset has 150M 100-bp pair end reads.

Lastly, we use STAR [1] (2-pass strategy) to align three simulated datasets onto the human reference genome (hg19)[3]. From each alignment (in bam format), we subsample to get smaller alignment files of chromosome15 as fixed datasets for BOAssembler. The small datasets are about 1.5%, 3.1%, and 2.1% of large datasets for HESC, LC and Kidney respectively. We've proposed another more complicated sampling method (available at Github site) across chromosomes, which offers similar performance.

3.2 Experiment Procedure

For each dataset, we run BOAssembler on the smaller datasets. The evaluation for metric also uses a subset (e.g. chromosome 15) of reference transcriptome. Each iteration takes around 1 min, and we typically see convergence of metric score around 40 to 50 iterations. Compared to grid search for possible combinations of 10 to 20 parameters, BOAssembler is much more efficient.

After automatic tuning, BOAssembler will recommend parameters with high metric scores. We then apply these parameters on large datasets, which typically take several hours to finish the assembly tasks using 25 cores of a linux server.

3.3 Experiment Results

Table 1 compares the performance of default parameters (Default) and BOAssembler-tuned parameters (Tuned) for each simulated dataset, in terms of sensitivity and precision. We also list their standard F1 score here since it's related to the metric BOAssembler tries to optimize. But we'll focus on sensitivity and precision which are of practical interest.

As Table 1 shows, BOAssembler has improved sensitivity, and precision for all small datasets. In particular, HESC small is improved by 16.9% in sensitivity and 27.4% in precision, LC small is improved by 1.2% in sensitivity and 13.1% in precision, Kidney small is improved by 3.2% in sensitivity and 5.5% in precision. Notice that the real Kidney dataset has been used in Stringtie's original work, so the default parameters of Stringtie should have been adjusted for this dataset statistics. Still BOAssembler improves the its performance further.

The trend of performance improvement is mostly reflected in assembly tasks on large datasets, which is mostly interesting to us. In particular, HESC large is improved by 6.5% in sensitivity and 32.2% in precision, Kidney large is improved by 0.3% in sensitivity and 0.6% in precision. LC large has a small loss around

[3] http://hgdownload.cse.ucsc.edu/goldenpath/hg19/bigZips/.

Table 1. Performance on different simulated datasets

Dataset	Sensitivity (%)		Precision (%)		F1	
	Default	Tuned	Default	Tuned	Default	Tuned
HESC (small)	22.1	39	31.9	59.3	26.11	47.05
HESC (large)	14.3	20.8	54.2	86.4	22.63	33.53
LC (small)	25.8	27	40	53.1	31.37	35.8
LC (large)	15.7	14.5	64.3	74.3	25.24	24.26
Kidney (small)	20.1	23.3	27.9	33.4	23.37	27.45
Kidney (large)	14.8	15.1	54.1	54.7	23.24	23.67

1.2% in sensitivity, but it gains 10%, which is significant, in precision. The experiments show that by tuning hyper-parameters through BOAssembler on small datasets, we are able to improve large assembly tasks overall (though there could be fluctuations) to a smaller extent.

The diminished performance gain of tuned parameters on large datasets, compared to the gain on small ones, may be because of an averaging effects across more variant alignment statistics in large datasets. To better catch up large dataset statistics, we have also prepared small datasets selected from certain regions, the performance improvement trend is similar.

By comparing the BOAssembler suggested parameters with assembler's default ones, we could also gain more insights into the datasets. For example, in HESC small datasets, the parameter 'f' is suggested to decrease from 0.1 to 0, this will allow more transcripts of low expression levels to also be considered as assembly output (hereby improve sensitivity). Meanwhile, the parameter 'm' is suggested to increase from 200 to 500 to allow only longer (e.g. at least 500) assembled transcripts to be considered (hereby improve precision).

3.4 Discussion

We expect our study and developed BOAssembler will contribute to the assembly community as follows: (a) For bioinformaticians who develop assembly algorithms, the framework or ideas behind it could provide them with more convenient ways to set default parameters for their assemblers. (b) For biologists who use reference-based RNA-Seq assemblers, BOAssembler can help them improve assembly performance, so they can gain better insights into the datasets, and the improved assembled RNA transcripts will be helpful for downstream gene, protein and cell related analysis. (c) For benchmark work of assemblers, typically several datasets are prepared and different assemblers are compared by using their default parameters. BOAssembler or its ideas will help the benchmark work in a fairer basis, since default parameters can not guarantee consistent good performance across various datasets.

Whereas this is, to our best knowledge, the first efforts to bring assembly and BO together, there are several interesting future directions. (a) As from

experiments, we have observed that the gain of tuned parameters gets diminished for larger datasets, which implies BO tuned parameter overfits to small training dataset. Since evaluating assembler is expensive, more efficient data subsampling and cross-validation methods to avoid overfitting will be helpful. (b) Another interesting exploration is how to define a metric score that is better than the current weighted F1 score for Bayesian Optimization, to better balance sensitivity and precision. (c) There're many problems in assembly areas (including variant calling) that heavily relay on hyper-parameter tuning for better performance. Introduce similar frameworks to these problems shall have wide applications.

Funding. This project is funded by NIH R01 Award 1R01HG008164 by NHGRI and NSF CCF Award 1703403.

References

1. Dobin, A., et al.: STAR: ultrafast universal RNA-seq aligner. Bioinformatics **29**(1), 15–21 (2013). https://academic.oup.com/bioinformatics/article/29/1/15/272537
2. Pertea, M., Pertea, G.M., Antonescu, C.M., Chang, T.-C., Mendell, J.T., Salzberg, S.L.: StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. Nat. Biotechnol. **33**, 290–295 (2015)
3. Trapnell, C., et al.: Transcript assembly and quantification by RNA-seq reveals unannotated transcripts and isoform switching during cell differentiation. Nat. Biotechnol. **28**(5), 511–515 (2010)
4. Grabherr, M.G., et al.: Full-length transcriptome assembly from RNA-seq data without a reference genome. Nat. Biotechnol. **29**(7), 644–652 (2011)
5. Kannan, S., Hui, J., Mazooji, K., Pachter, L., Tse, D.: Shannon: an information-optimal de Novo RNA-seq assembler. bioRxiv (2016)
6. Hayer, K.E., Pizarro, A., Lahens, N.F., Hogenesch, J.B., Grant, G.R.: Benchmark analysis of algorithms for determining and quantifying full-length mRNA splice forms from RNA-seq data. Bioinformatics **31**, 3938–3945 (2015). https://doi.org/10.1093/bioinformatics/btv488
7. Kececioglu, J.D., Myers, E.W.: Combinatorial algorithms for DNA sequence assembly. Algorithmica **13**(1–2), 7–51 (1995)
8. Frazier, P.I.: A tutorial on Bayesian optimization. arXiv preprint arXiv:1807.02811, 8 July 2018
9. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**(Feb), 281–305 (2012)
10. Snoek, J., et al.: Scalable Bayesian optimization using deep neural networks. In: International Conference on Machine Learning, pp. 2171–2180, 1 June 2015
11. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2, NIPS 2012, USA, pp. 2951–2959. Curran Associates Inc. (2012)
12. Frazier, P.I.: A tutorial on Bayesian optimization (2018)
13. Wang, L., Feng, M., Zhou, B., Xiang, B., Mahadevan, S.: Efficient hyper-parameter optimization for NLP applications. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 2112–2117 (2015)

14. Brochu, E., Cora, V.M., De Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599, 12 December 2010
15. Jiang, Y., Kim, H., Asnani, H., Kannan, S., Oh, S., Viswanath, P.: LEARN codes: inventing low-latency codes via recurrent neural networks. arXiv preprint arXiv:1811.12707, 30 November 2018
16. The GPyOpt authors. GPyOpt: A Bayesian optimization framework in python (2016). http://github.com/SheffieldML/GPyOpt
17. Quitadadmo, A., Johnson, J., Shi, X.: Bayesian hyperparameter optimization for machine learning based eQTL analysis. In: Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics - ACM-BCB 2017. ACM Press (2017)
18. Au, K.F., et al.: Characterization of the human ESC transcriptome by hybrid sequencing. Proc. Natl. Acad. Sci. U.S.A. **110**(50), E4821–4830 (2013)
19. Tilgner, H., Grubert, F., Sharon, D., Snyder, M.P.: Defining a personal, allele-specific, and single-molecule long-read transcriptome. Proc. Natl. Acad. Sci. U.S.A. **111**(27), 9869–9874 (2014)
20. Li, B., Dewey, C.N.: RSEM: accurate transcript quantification from RNA-seq data with or without a reference genome. BMC Bioinform. **12**(1), 323 (2011)