RESEARCH ARTICLE

BinPacker: Packing-Based *De Novo* Transcriptome Assembly from RNA-seq Data

Juntao Liu[1]☯, Guojun Li[1]☯*, Zheng Chang[1], Ting Yu[1], Bingqiang Liu[1], Rick McMullen[2], Pengyin Chen[3], Xiuzhen Huang[4]*

1 School of Mathematics, Shandong University, Jinan, China, **2** High Performance Computing Center, University of Arkansas, Fayetteville, Arkansas, United States of America, **3** Crop, Soil, and Environmental Sciences, University of Arkansas, Fayetteville, Arkansas, United States of America, **4** Department of Computer Science, Arkansas State University, Jonesboro, Arkansas, United States of America

☯ These authors contributed equally to this work.
* guojunsdu@gmail.com (GL); xhuang@astate.edu (XH)

OPEN ACCESS

Citation: Liu J, Li G, Chang Z, Yu T, Liu B, McMullen R, et al. (2016) BinPacker: Packing-Based *De Novo* Transcriptome Assembly from RNA-seq Data. PLoS Comput Biol 12(2): e1004772. doi:10.1371/journal.pcbi.1004772

Editor: Thomas Lengauer, Max-Planck-Institut für Informatik, GERMANY

Received: March 19, 2015

Accepted: January 18, 2016

Published: February 19, 2016

Data Availability Statement: All relevant data are within the paper and its Supporting Information files.

Funding: This work was funded by National Natural Science Foundation of China with codes 61432010, 61272016 and 31571354 (http://www.nsfc.gov.cn/); National Science Foundation with number 1553680 (http://www.nsf.gov/); National Center for Research Resources with number P20RR016460 (http://sdminutes.cit.nih.gov/about/almanac/organization/NCRR.htm); and National Institute of General Medical Sciences with number P20GM103429 (http://www.nigms.nih.gov/Pages/default.aspx). The funders had

Abstract

High-throughput RNA-seq technology has provided an unprecedented opportunity to reveal the very complex structures of transcriptomes. However, it is an important and highly challenging task to assemble vast amounts of short RNA-seq reads into transcriptomes with alternative splicing isoforms. In this study, we present a novel *de novo* assembler, BinPacker, by modeling the transcriptome assembly problem as tracking a set of trajectories of items with their sizes representing coverage of their corresponding isoforms by solving a series of bin-packing problems. This approach, which subtly integrates coverage information into the procedure, has two exclusive features: 1) only splicing junctions are involved in the assembling procedure; 2) massive pell-mell reads are assembled seemingly by moving a comb along junction edges on a splicing graph. Being tested on both real and simulated RNA-seq datasets, it outperforms almost all the existing *de novo* assemblers on all the tested datasets, and even outperforms those *ab initio* assemblers on the real dog dataset. In addition, it runs substantially faster and requires less memory space than most of the assemblers. BinPacker is published under GNU GENERAL PUBLIC LICENSE and the source is available from: http://sourceforge.net/projects/transcriptomeassembly/files/BinPacker_1.0.tar.gz/download. Quick installation version is available from: http://sourceforge.net/projects/transcriptomeassembly/files/BinPacker_binary.tar.gz/download.

Author Summary

The availability of RNA-seq technology drives the development of algorithms for transcriptome assembly from very short RNA sequences. However, the problem of how to (*de novo*) assemble transcriptome using RNA-seq datasets has not been modeled well; e.g. sequence coverage information has even not been accurately and effectively integrated into the appropriate assembling procedure, leading to a bottleneck that all the existing (*de novo*) strategies have encountered. We present a novel approach to remodel the problem as tracking a set of trajectories of items with their sizes representing the coverage of their

no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The authors have declared that no competing interests exist.

corresponding isoforms by solving a series of bin-packing problems. This approach, which subtly integrates the coverage information into the procedure, has two exclusive features: 1) only splicing junctions are involved in the assembling procedure; 2) massive pell-mell reads are assembled seemingly by moving a comb along junction edges on a splicing graph. Being tested on both real and simulated RNA-seq datasets, it outperforms almost all existing *de novo* assemblers on all the tested datasets, even outperforms those *ab initio* assemblers on the dog dataset, in terms of commonly used comparison standards.

This is a *PLOS Computational Biology* Methods paper.

Introduction

The advent of RNA-seq techniques are changing how transcription, splicing variations and associated mechanisms can be studied since they provide unprecedented accuracy about the mRNA expression level [1]. They allow accurate elucidation of all splicing variants, including the rare and lowly expressed splicing isoforms. This clearly opens many new doors for studying the mechanisms of various human diseases that are related to abnormal splicing [1], including cancers. With the RNA-seq techniques, there come new challenges associated with the interpretation of the generated datasets. Although sequencing reads from PacBio RS II sequencer are long enough to cover multiple exons, they have not been commonly used to improve the state of the art transcripts reconstruction because they are suffering from higher error rates [2]. Therefore the RNA-seq techniques for short sequencing reads [3] remain necessary. One major challenge is how to accurately assemble the short sequencing reads into full-length transcripts possibly involving multiple splicing variants, the so-called RNA-seq based transcriptome assembly problem.

According to the literatures [4–6], there are various alternative splicing events capable of producing multiple isoforms in eukaryotic genes. Event types include skipped exons, retained introns and mutually exclusive exons. Even more complicated, some exons may be partially involved in transcripts during the alternative splicing process. At first glance, the transcriptome assembly is similar to genome assembly, but they are actually fundamentally different. In contrast, the following facts make the transcriptome assembly more challenging: (i) some transcripts have a very low expression level, while others may be expressed in a dramatically high level [7]; (ii) each locus usually produces multiple transcripts due to various alternative splicing events [8]; (iii) some transcripts with low expression level may be submerged due to the sequencing errors [8,9]. Therefore, a successful transcriptome assembler should overcome all these difficulties, and be capable of recovering all full-length transcripts of variable lengths, expression levels and noises.

Computational strategies for transcriptome assembly can be generally divided into two categories, *ab initio* and *de novo* [1,8]. If a reference genome is available, *ab initio* approaches, such as Cufflinks [10] and Scripture [11], usually start by mapping RNA-Seq reads to the reference genome, and then sequences with overlapping alignment are merged into a connectivity graph on which the well studied min-cost minimum path cover model is subtly employed to extract a minimum set of paths which explain the RNA-seq dataset. A very recently published *ab initio* assembler, StringTie [12], also first maps RNA-Seq reads to the reference genome, then constructs alternative splicing graphs and then assembles transcripts by using a maximum-flow network model. *De novo* approaches, such as ABySS [13], SOAPdenovo-Trans [14], Oases [15] and IDBA-Tran [16], directly use the reads to assemble transcripts, without mapping them to a reference genome, which is important when the reference genome is unavailable, incomplete, highly fragmented or substantially altered as in cancer tissues. These *de novo* approaches which were developed based on the techniques used in genome assembly are not solving all the transcriptome assembly problems in general [7]. Trinity [8] which was designed specifically for *de novo* transcriptome assembly has substantially improved the state of the art *de novo* transcriptome assemblers. It starts by extending short reads through overlaps into contigs, connecting contigs into a graph, and then extracts paths from this graph to construct splicing variants based on a brute-force enumeration strategy. Trinity does improve previous *de novo* assemblers which have their roots in genome assembly techniques, but it does not introduce an appropriate model to optimize its solution, and even not incorporate sequencing coverage depth information into the assembly procedure either, although the authors in Trinity have noticed that similarity of the coverage depth across different coding regions in a transcript could be useful. To this end, we have recently presented a new *de novo* transcriptome assembler, Bridger [17], which "bridges" between Cufflinks and Trinity so that the techniques used in Cufflinks can be employed to overcome the limitations of Trinity. Bridger does incorporate the coverage information into the assembly procedure via an appropriate model, but it could not guarantee a genuine solution due to (1) in-weight and out-weight are defined somewhat arbitrarily in Bridger; (2) a node with both in-edges and out-edges has no chance to be an end of any transcripts. Therefore, there still remains room for improvement.

In this paper we develop a novel *de novo* algorithm, BinPacker, to assemble full-length transcripts by remodeling the problem as tracking a set of trajectories of items over a splicing graph, which is constructed by employing the techniques used in Bridger [17] with several updates described in Methods. The set of trajectories of items over the splicing graph can be achieved by solving a series of variants of the bin-packing problem, which are different from the traditional bin-packing problem, which is defined to pack a given number of items of different sizes into as few bins of a given size as possible, and each bin can only hold items with the sum of their sizes no more than the size of the bin. We have tested and compared BinPacker with seven competitive *de novo* assemblers, Trinity [8], ABySS [13], Trans-ABySS [18], SOAPdenovo-Trans [14], Oases [15], IDBA-Tran [16] and Bridger [17] on real and simulated datasets. The simulation dataset is generated as described in Results section. For the real datasets, three datasets are used, including two standard RNA-seq datasets, one dog and one human, and one strand-specific mouse RNA-seq dataset. The comparison results show that BinPacker outperforms almost all the compared assemblers on all datasets, including real and simulated, in terms of commonly used standards for evaluation of transcriptome assemblers. Even more surprisingly, it outperforms StringTie, a most recently published *ab initio* assembler [12], on dog dataset.

Results

We ran BinPacker, and seven other *de novo* assemblers: ABySS (version 1.3.4), Trans-ABySS (version 1.4.4), Trinity (version 2012-10-05), Velvet (version 1.2.01) + Oases (version 0.2.02), SOAPdenovo-Trans (version 1.01), IDBA-Tran (version 1.1.1), and Bridger, and also the most recently published *ab initio* assembler StringTie on real and simulated datasets below, and tested their performance with the optimized parameters on the same server with 512GB of RAM (see S1 Text for details).

The criteria that have previously been used to test the assemblers are employed in our testing. All assembled transcripts are matched against all known transcripts in the annotation (referred to as "reference transcripts") using BLAT [19], with 95% identity as the cutoff. If an assembled transcript full-length covers a reference transcript with at least 95% sequence identity and at most 0.5% indels, this reference transcript is counted as full-length recovered, and noted as a true positive. The indel cutoff is used to avoid the over-estimated consistencies between the predicted and the references. In this paper, sensitivity is defined as the number of full-length recovered reference transcripts, and accuracy is defined as the true positive rate. We further consider two types of accuracy. One is related to a reference true positive rate which is the rate between the number of full-length recovered reference transcripts and the number of assembled transcripts, and the other is related to an assembled true positive rate which is the fraction of assembled transcripts that are in the reference transcripts. The reliability of an assembler is defined by the distribution of its recovered reference transcripts against recovered sequence length rates ranging from 80% to 100%. An assembler is considered of higher reliability if it recovers more reference transcripts with recovered sequence length rates ranging from 90% to 100%.

1. Tests on real datasets

We ran and tested all the 9 assemblers on three real RNA-seq datasets which include two standard (non-strand specific) Illumina datasets from dog and human, and one strand-specific dataset from mouse.

1.1. Collection of real datasets. The dog dataset was collected from NCBI SRA database (Accession Code: SRR882093), the human dataset was collected from the DDBJ SRA database (Accession Codes: SRX011545 and SRX011546) and the mouse dataset was collected from C567BL/6 mouse primary immune dendritic cells (Accession Code: SRX062280 in the DDBJ SRA database). The reference transcripts of dog were downloaded from UCSC [20]. The human and mouse reference transcripts were downloaded from Ensemble Genome Browser [21].

1.2. Comparing BinPacker to the other assemblers on real datasets. We compare BinPacker to the other assemblers on the real datasets mentioned above in terms of sensitivities, accuracies and their distributions against recovered sequence length rates ranging from 80% to 100%.

1.2.1 Comparison of sensitivities and their distributions against recovered sequence length rates on dog and mouse datasets. We run all the *de novo* assemblers on dog and mouse datasets. The results show that BinPacker reaches the highest sensitivity, recovering 1149 and 10012 full-length transcripts among 33665 and 39060 candidates respectively on dog and mouse datasets, while Trinity recovers 1091 and 9599 among 49311 and 78333, and Bridger recovers 1147 and 9991 among 37234 and 50051. Bridger performs a little worse than BinPacker, but better than Trinity and all the other *de novo* assemblers (Fig 1A and 1C, shaded area). Trinity performs worse than BinPacker because it uses an exhaustive enumeration algorithm to search for paths in *de Bruijn* graphs without using sequencing depth information in the searching process, which results in the increase of false positives and the decrease of true positives. Bridger performing worse than BinPacker is due to the facts: 1) the weights in compatibility graph are defined a bit arbitrarily, and 2) a node with both in-edges and out-edges in the splicing graph will never be an end of a transcript. Apart from the three best *de novo* assemblers mentioned above, Trans-ABySS performs best on dog dataset, while Oases does best on mouse dataset. We further compared BinPacker with StringTie, a most recently published *ab initio* assembler. As expected, StringTie performs best on mouse dataset. Surprisingly, while it is defeated by BinPacker on dog dataset, StringTie recovers 1072 full-length transcripts, compared to 1149 recovered by BinPacker.

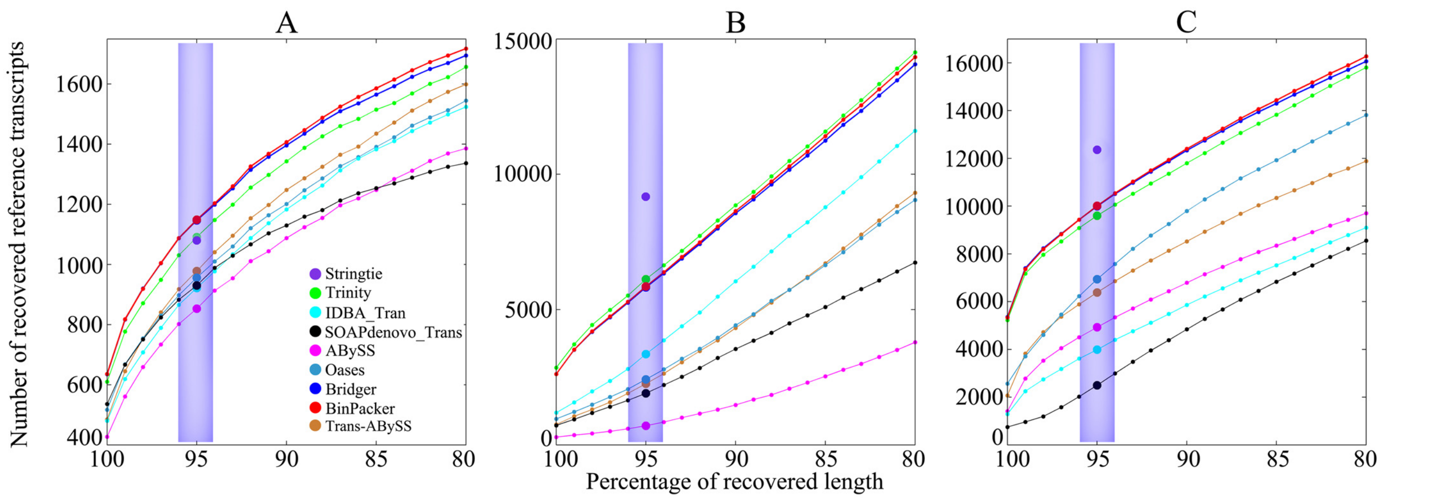

Fig 1. Comparison of recovered reference sensitivity and its distribution against recovered sequence length rates (sequence identity) ranging from 80% to 100% on (A) dog, (B) human and (C) mouse datasets. Solid colored circles in shaded areas represent the number of full-length recovered reference transcripts for different assemblers.

doi:10.1371/journal.pcbi.1004772.g001

To test the reliability of these *de novo* assemblers, we compare their sensitivity distributions against recovered sequence length rates ranging from 80% to 100%. As shown in Fig 1A and 1C, BinPacker keeps the highest sensitivity in the whole interval [80%, 100%] on both dog and mouse datasets. Bridger's sensitivity is a little lower than BinPacker's, while Trinity is lower than both BinPacker and Bridger, but higher than the others in the whole interval [80%, 100%].

1.2.2 Comparison of accuracies and distributions against recovered sequence length rates on dog and mouse datasets. Our comparison results show that BinPacker outperforms all the other *de novo* assemblers we are comparing with in terms of both types of accuracy on dog and mouse datasets (Figs 2A and 2C and 3A and 3C, shaded area). Of all the other assemblers, Bridger performs best on dog dataset in terms of both types of accuracy, while ABySS performs best on mouse dataset in terms of both types of accuracy. Trinity suffers from very low accuracy on both dog and mouse datasets because of its large false positives. StringTie as expected performs best on mouse dataset, but unexpectedly worse than BinPacker on the dog dataset in terms of both types of accuracy.

The comparison results of accuracy distributions against recovered sequence length rates ranging from 80% to 100% (Figs 2A and 2C and 3A and 3C) show that BinPacker keeps the highest accuracy level in the interval [90%, 100%] on both dog and mouse datasets. The following are some details of the performances of the other assemblers excluding BinPacker. Bridger keeps the highest accuracy level among the others on dog dataset in terms of both types of accuracy in the interval [90%, 100%]. On the mouse dataset, ABySS keeps the highest among the others excluding Bridger in terms of reference true positive rate in the interval [90%, 100%], and Oases keeps the highest among the others excluding Bridger in terms of assembled true positive rate in the interval [80%, 100%] excluding [94%, 99%]. Trinity again loses in accuracy of both types in the interval [80%, 100%] on both dog and mouse datasets.

Therefore we conclude that BinPacker has the highest reliability among all the *de novo* assemblers we are comparing with in terms of their distributions of sensitivity and accuracy against recovered sequence length rates on real dog and mouse datasets.

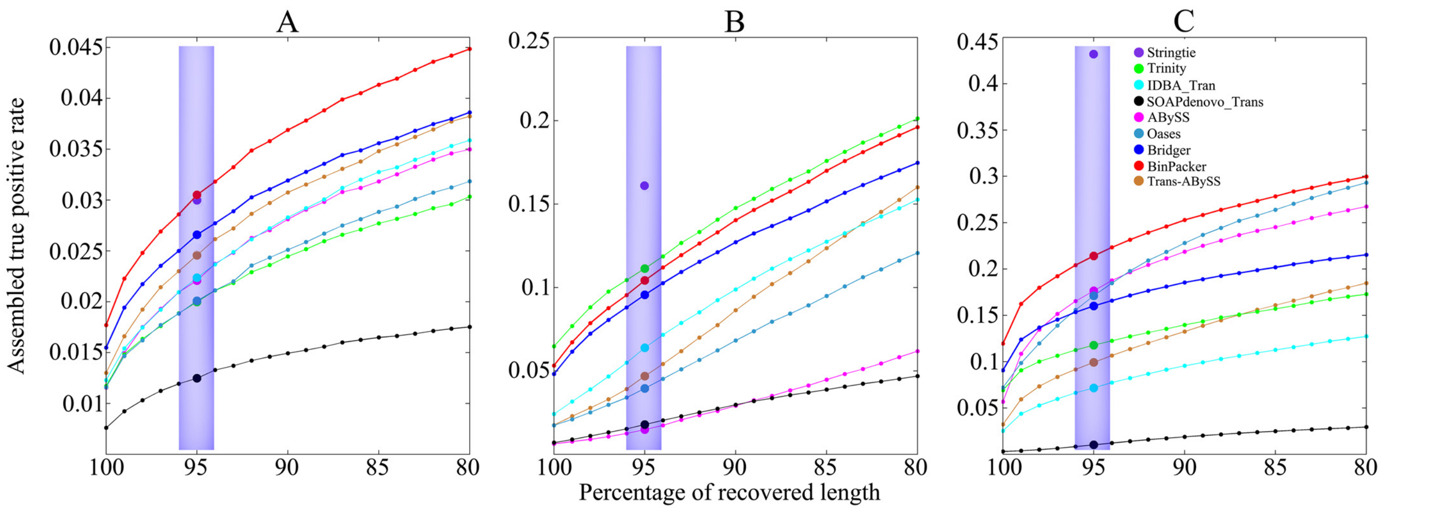

Fig 2. Comparison of assembled true positive rate and its distribution against recovered sequence length rates (sequence identity) ranging from 80% to 100% on (A) dog, (B) human and (C) mouse datasets. Solid colored circles in shaded areas represent the assembled true positive rate for different assemblers.

doi:10.1371/journal.pcbi.1004772.g002

1.2.3 BinPacker maintains a stable performance on human dataset. The human dataset is also adopted to test the performance of BinPacker and the other assemblers. The results show that BinPacker outperforms all the other *de novo* assemblers except Trinity on some cases. The following are some details. For the sensitivity, BinPacker, Bridger and Trinity recovered 5859, 5822 and 6122 full-length reference transcripts from 41691, 41470 and 54315 candidates, respectively. StringTie recovered 9177 full-length reference transcripts out of 43757 candidates. Again Trinity gets more false positives than BinPacker and Bridger. The reference true positive rate of BinPacker is 14.05%, best of all *de novo* assemblers, while that of Trinity is 11.27%, higher than all the other assemblers except Bridger, which performs only a little worse than BinPacker in this measure, with its reference true positive rate 14.03%; the assembled true positive rate of BinPacker achieves 10.37%, while Trinity reaches 11.14%, highest among the

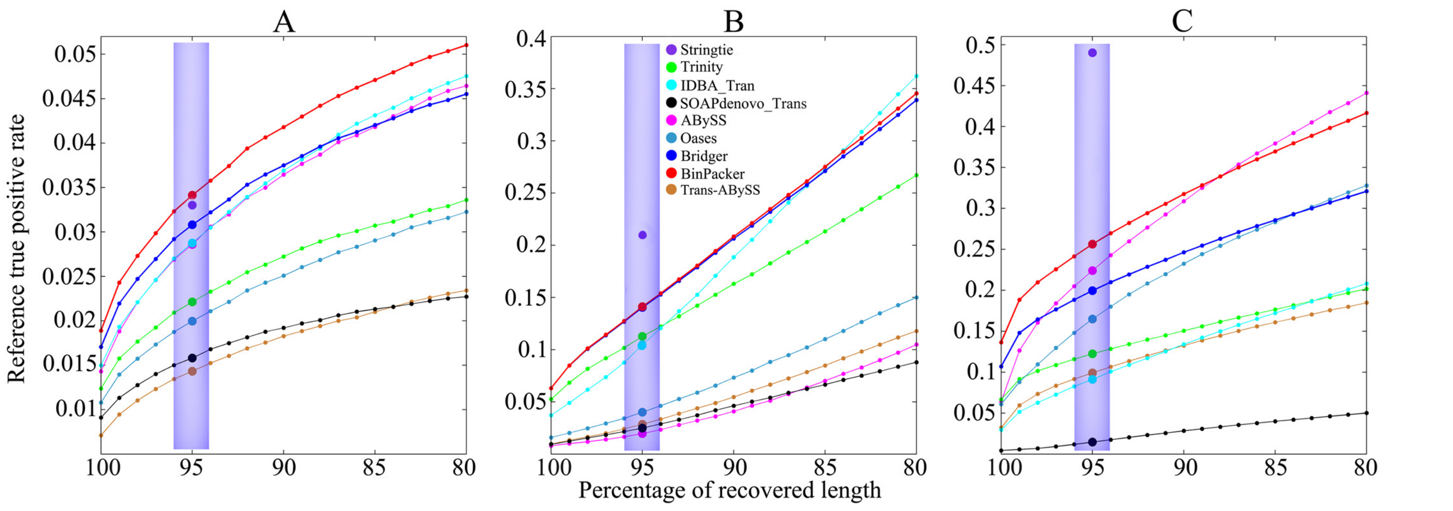

Fig 3. Comparison of reference true positive rate and its distribution against recovered sequence length rates (sequence identity) ranging from 80% to 100% on (A) dog, (B) human and (C) mouse datasets. Solid colored circles in shaded areas represent the reference true positive rate for different assemblers.

doi:10.1371/journal.pcbi.1004772.g003

compared *de novo* assemblers, including Bridger, with its assembled true positive rate 9.56%. We also compute the sensitivity and accuracy distributions against recovered sequence length rates ranging from 80% to 100%. For the sensitivity distribution, the three curves of BinPacker, Bridger and Trinity are almost coincident with the highest sensitivity among all *de novo* assemblers. For the accuracy, the reference true positive rate of BinPacker keeps the highest in the interval [90%, 100%]. For the assembled true positive rate, BinPacker performs a little worse than Trinity, which reaches the highest in the whole interval, but much better than the others. The performance of the other assemblers on human dataset is almost the same as on dog and mouse datasets (See Figs 1B–3B for details).

2. Tests on simulated dataset

It is necessary to test the assemblers using simulated RNA-seq dataset since we may know all the genuine transcripts hidden in it in advance. An *in silico* RNA-Seq data generator, Flux Simulator [22], is applied to UCSC hg19 gene annotation to generate an error-free dataset of approximately 50 million paired-end strand-specific RNA-seq reads. To demonstrate the advantage of BinPacker over other assemblers on the simulated dataset, we ran all the assemblers and did comparison among them in terms of their sensitivities, accuracies and their distributions against recovered sequence length rates.

Our comparison results show that BinPacker not only reaches the highest sensitivity, but also the highest accuracy levels of both types. Furthermore, BinPacker keeps the highest sensitivity and accuracy of both types in the whole interval [80%, 100%]. Therefore it can be concluded that BinPacker has the highest reliability among all the *de novo* assemblers we are comparing with in terms of their distributions of both sensitivity and accuracy against recovered sequence length rates on the simulated dataset. See Fig 4 and S1 Text for details.

3. Comparison of running time and memory usage on real datasets

We examined the computing resources required by these *de novo* assemblers: the running time and the memory usage on the same server. The results are shown in Figs 5 and 6. ABySS uses

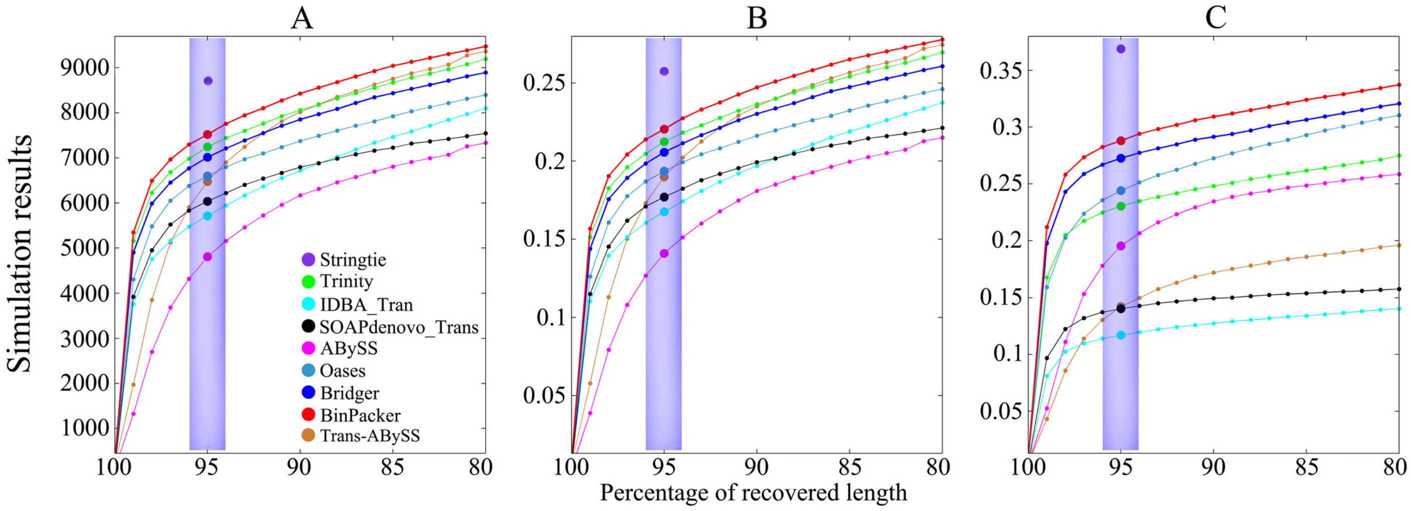

Fig 4. Comparison of assemblers on simulated dataset. (A) Recovered reference sensitivity and its distribution against recovered sequence length rates. The solid colored circles in shaded areas represent the number of full-length recovered reference transcripts for different assemblers; (B) Reference true positive rate and its distribution against recovered sequence length rates. The solid colored circles in shaded areas represent the reference true positive rate for different assemblers; (C) Assembled true positive rate and its distribution against recovered sequence length rates. The solid colored circles in shaded areas represent the assembled true positive rate for different assemblers.

doi:10.1371/journal.pcbi.1004772.g004

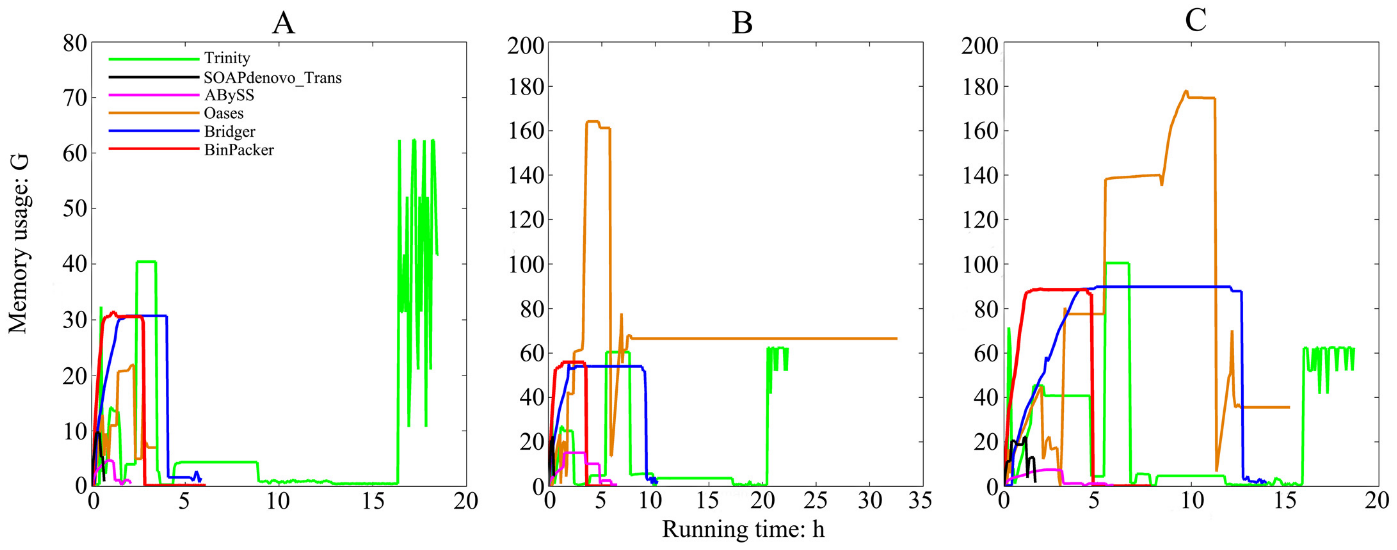

Fig 5. RAM usage for each assembler on (A) dog, (B) human and (C) mouse datasets. Same parameter values are used for all assemblers: k = 25 and CPU = 6.

doi:10.1371/journal.pcbi.1004772.g005

the least memory (Fig 5), while SOAPdenovo-Trans takes the shortest time (Fig 6). Oases performs well on dog dataset but it consumes the most memory and has almost the longest running time on both human and mouse dataset. We noted that the computing resource requirement by Oases is sensitive to the *k*-mer value, which had also been found in another research paper [23]. As an enumeration algorithm, Trinity consumes the most memory on dog dataset and takes the longest time on both dog and mouse datasets. For the memory usage (Fig 5) BinPacker and Bridger almost require the same amount of memory, more than most of the compared assemblers except Trinity and Oases, which consume much more memory than Bin-Packer on human and mouse datasets. For the time usage (Fig 6), BinPacker is among the fastest assemblers and it has also made a great improvement compared to Bridger, which takes much more time than BinPacker on both human and mouse datasets.

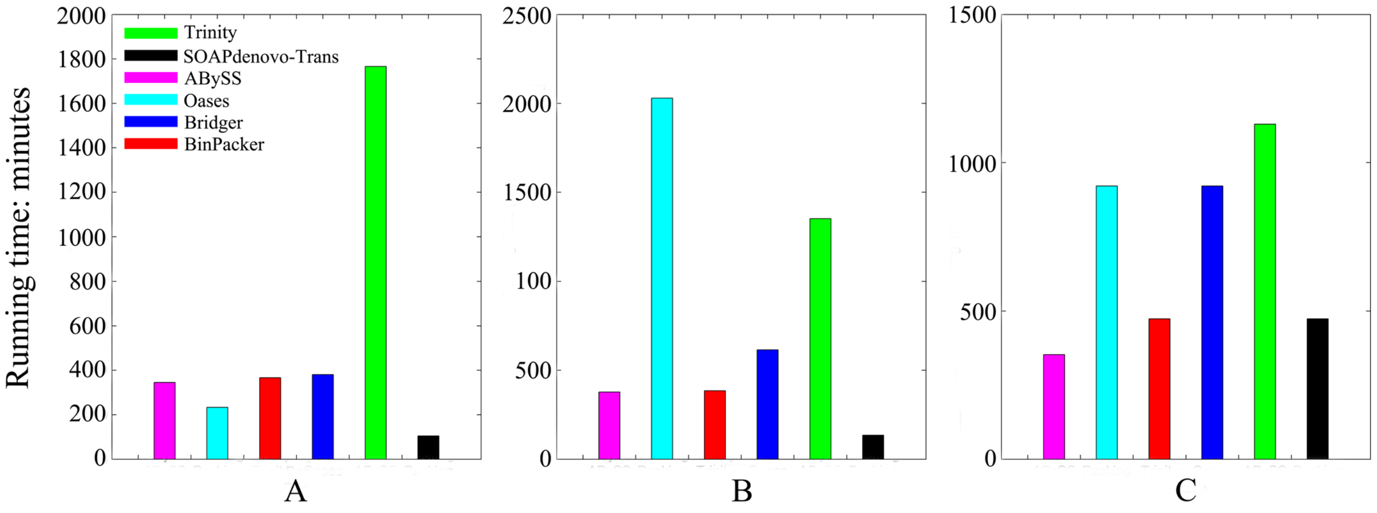

Fig 6. Running time for each assembler on (A) dog, (B) human and (C) mouse datasets.

doi:10.1371/journal.pcbi.1004772.g006

Discussion

In this study, we presented a novel *de novo* method BinPacker for transcriptome assembly using short RNA-seq reads. Compared with Trinity, one of the most popular *de novo* assemblers, BinPacker has the following advantages: (i) Trinity uses a fixed k-mer length 25, which is not necessarily optimal for all datasets, while BinPacker allows different user-specified k-mer values for different problems for optimal performance. One crucial parameter of BinPacker is the k-mer length. Generally speaking, with larger k values it performs better on high expression datasets or longer reads and with smaller k values it performs better on low expression datasets or shorter reads [17]. In light of our testing results, k = 25 is chosen to be the default value, however, larger k values are recommended for reads with length longer than 75bp. (ii) Compared to the exhaustive enumeration method used in Trinity, BinPacker uses a rigorous mathematical model to search for an optimal set of paths from the splicing graph, which makes BinPacker achieve a lower false positive rate at the same level of sensitivity. (iii) BinPacker makes full use of the sequencing depth information, which is applied to define the junction weights of the splicing graphs, constraining the deconvolution of splicing graphs into individual transcripts, and hence making its assembly results more accurate. (iv) BinPacker makes a different use of the paired-end information compared with Trinity. While Trinity uses the paired-end information to search for paths in the *de Bruijn* graphs, this information is mainly used in our process of constructing splicing graphs. Firstly, the paired-end information is used to help reconstruct more complete splicing graphs, making contigs even not covered by overlapping k-mers be recovered during assembly. Secondly, paired-end information is also used to trim error branches of the constructed splicing graphs, removing error junctions from splicing graphs. In practice, BinPacker uses less memory space and shorter running time.

As showed in Results section, the assemblers have a high variance in sensitivity, accuracy and time and memory usage across the different RNA-seq datasets. Several facts may cause such a variance. 1) Different RNA-Seq datasets may contain different transcripts expression levels and different sequencing depths, both of which lead to the same transcripts in different RNA-Seq datasets covered by quite a different number of reads. And so they could have a large effect on sensitivity, accuracy and time and memory usage. 2) The reads in different RNA-Seq datasets may have different lengths, maybe shorter than 50, and maybe longer than 100, which may also cause differences in sensitivity, accuracy and time and memory usage. 3) The qualities of reference transcripts for different species are also quite different. For example, human and mouse genomes have been studied more extensively than dog genome, so the rate of known reference transcripts will certainly be larger than that of dog. We have seen in our comparison the sensitivity and accuracy of dog is lower than that of human and mouse. 4) Other reasons, such as different sequencing error rates, the usage of paired-end reads or single-end reads, may also contribute to the variance in sensitivity, accuracy and time and memory usage.

The E. coli dataset is also adopted to evaluate the performance of the *de novo* assemblers on low complexity genome species without alternative splicing isoforms. Since the dataset is much smaller than that of dog, human and mouse, all the compared assemblers use much less running time and memory usage. For the sensitivity and accuracy, because most compared assemblers are designed to assemble transcripts from genes with alternative splicing events, they all do not perform very well on low complexity genome species such as E. coli without alternative splicing isoforms. Details are described in the first section of the S1 Text.

As far as we know, BinPacker is the first algorithm using the bin-packing strategy for *de novo* assembly, without the utilization of any other reference information. Tested on both real and simulated RNA-seq datasets, BinPacker shows the best sensitivity and accuracy compared to all the other *de novo* assemblers, and even outperforms the most popular *ab initio* assembler

StringTie on real dog dataset, only slightly worse than Trinity in some aspects on real human dataset. In addition, it requires fewer computational resources and less running time compared to most of the other assemblers. With these demonstrated advantages, we anticipate that BinPacker will play an important role for new discoveries in transcriptome study using RNA-seq datasets, especially for cancer transcriptomic data analyses.

Methods

The splicing graph was first introduced by Heber, et al in 2002 [24]. BinPacker assembles transcripts on each splicing graph it constructs. Each splicing graph constructed by BinPacker is a directed acyclic graph, with its nodes and edges representing exons and splicing events of the gene. The nodes in the splicing graph are continuous genome sequences without any alternative splicing events, which may not be real exons of the gene. Based on the generalized definition of exons, BinPacker first builds splicing graphs for all expressed genes encoded in the genome using the given RNA-seq datasets. Ideally, each splicing graph constructed by BinPacker has a correspondence to a specific (expressed) gene. Unfortunately, it may not always be this case due to the existence of sequencing errors, homologous genes and low expression levels of some genes. But it does not cause a serious impact on our full-length transcripts recovery of individual genes even though some splicing graphs cover multiple genes or only parts of a gene. Without loss of generality, we assume that each splicing graph represents one expressed gene. Having constructed all the splicing graphs, BinPacker searches for an optimal edge-path-cover over all the individual splicing graphs by iteratively solving a series of bin packing problems. Each edge-path-cover output by BinPacker can explain all the observed splicing events encoded in the corresponding splicing graph. A flowchart of the BinPacker algorithm is given in Fig C in S1 Text.

1. Construction of splicing graphs

BinPacker constructs splicing graphs based on the method of Bridger [17] with several updates as follows. First of all, while Bridger is not able to process RNA-Seq reads with different lengths, BinPacker can handle reads with variable lengths. Secondly, Bridger trims the branches of the splicing graphs after all splicing graphs have been constructed. However, BinPacker trims splicing graphs during the construction of splicing graphs.

2. Topological ordering of splicing graph and detecting a maximal set of pairwise incompatible edges

Two directed edges in a splicing graph are said to be compatible if they may come from one directed path, and incompatible otherwise (see Fig I in S1 Text). We may imagine that the splicing graphs one-to-one correspond to the expressed genes, with nodes corresponding to exons and edges corresponding to splicing junctions. Since exons are linearly arranged in a gene, we may suppose that the nodes in the splicing graph of the gene are also linearly arranged, but not necessarily to be identical to that of the gene. We did this linearly arrangement by topological ordering of the splicing graph, which can be solved in linear time [25]. After topological ordering, all nodes with only out-edges are moved to the leftmost of the graph and all nodes with only in-edges to the rightmost. From now on, we refer to the splicing graph with all nodes being linearly arranged as a canonical splicing graph. Note that each directed edge in the canonical splicing graph can only go in the direction of the gene (Fig I in S1 Text). Each edge in a splicing graph is assigned a weight using its sequencing depth (number of reads spanning the junction edge in the splicing graph). It is obvious that the edges crossing two consecutive nodes in the splicing graph are pairwise incompatible (Fig J in S1 Text). In fact, the maximum set of edges crossing two consecutive nodes in a canonical splicing graph must be a maximal set of pairwise incompatible edges in the splicing graph (see Theorem 1 in S1 Text). BinPacker will iteratively execute a series of bin packing programs based on such a maximal set of pairwise incompatible edges.

3. Bin packing

BinPacker iteratively calls a variant of bin packing model to comb all the transcripts encoded in a splicing graph. To do so, we add a source node s and a sink node t into the splicing graph (Fig I in S1 Text), and connect s to the nodes with only out-going edges, and connect all the nodes with only in-coming edges to t. The weight of the new edge connecting s and u is assigned to be the sum of the weights of the edges going out from u. Similarly, the new edges going to t can be weighted.

Step 1: Balancing splicing graphs. Let u be a node in a splicing graph, the sum of the weights of the in-edges of u is said to be in-weight of u, denoted by $w_{in}(u)$. Out-weight of u is defined similarly, denoted by $w_{out}(u)$. Let $w_{min} = \min\{w_{in}(u), w_{out}(u)\}$, $c = \alpha(\gamma—\beta)/w_{min} + \beta$, where α, β and γ are parameters that users can specify (see Methods and Fig M in S1 Text for details). When there is a significant difference between $w_{in}(u)$ and $w_{out}(u)$, the node u is supposed to be an end of a transcript. We handle this by adding a new edge from the source s to the node u, with weight $w_{out}(u)–w_{in}(u)$ whenever $w_{out}(u)/w_{in}(u) \geq c$, which means that the difference between $win(u)$ and $w_{out}(u)$ is significant. Similarly, we may add a new edge from the node u to the sink t if $w_{in}(u)/w_{out}(u) \geq c$. BinPacker sets $\alpha = 10$, $\beta = 1.4$ and $\gamma = 1.5$ as default.

Step 2: Iterations of the bin packing. Suppose that we have a maximal set I of pairwise incompatible edges crossing the two consecutive nodes obtained above. BinPacker identifies each edge in the splicing graph as a bin with its capacity being the weight (sequencing depth) of the edge, and puts an item i in each bin (edge) in I. The size of the item i, denoted by w_i, is simply the weight of the edge (bin) where the item i resides. During the execution, BinPacker always faces a bin packing problem, which is slightly different from the traditional bin packing model. In our model, each item must be packed into one and only one bin and each bin can hold several items with the sum of their sizes smaller or larger than or equal to the capacity of the bin. At the very beginning, all the $|I|$ items are one-to-one put in the $|I|$ bins accordingly. Let n_L denote the left one of the two consecutive nodes and n_R the right one.

Starting from n_L, BinPacker carries out the first iteration of the bin packing. The first instance of bin packing towards left is formed as follows: we have as input the $|I|$ items defined from the edges in I, and a set I' of bins (edges) crossing the two consecutive nodes $n_L\text{-}1$ and n_L. What we are going to do is to optimally pack the $|I|$ items into the $|I'|$ bins. For the (heuristic) algorithm design, we partition the edges in I U I' into three sets I_{in}, I_{out} and I_m, with I_{in} consisting of the edges coming to n_L, I_{out} of the edges going out of n_L, and I_m of the remaining edges. Clearly, edges in I_{in} belong to I' but not to I, edges in I_{out} belong to I but not to I', and edges in I_m belong to both I and I'. For the first instance, we have that $|I_{out}|\geq|I_{in}|$ since $|I|\geq|I'|$. Executing the bin packing, BinPacker keeps the items in the bins (edges) of I_m staying unchanged, and optimally packs the items in the bins (edges) of I_{out} into the bins (edges) of I_{in} whenever $|I_{out}|\geq|I_{in}|$, then reset $n_L = n_L\text{-}1$, I_{in} and I_{out} accordingly. Repeat this procedure until encountering a *trap node* n_L with $|I_{out}|<|I_{in}|$, which may happen in two cases (see Methods in S1 Text), or reaching the source node s. If the former occurs in some iteration, BinPacker replaces the n (= $|I_{out}|$) items in the bins (edges) of I_{out} by m (= $|I_{in}|$) new items with sizes, $w_1, w_2,. . ., w_m$, the weights of the m edges of I_{in}, while the other items in the bins (edges) of I_m stay unchanged, and then executes the next iteration starting from the trap node n_L towards the opposite direction until encountering another trap node or reaching the sink node t; otherwise, the latter occurs, BinPacker jumps back to the starting node of the current iteration and processes the remaining nodes one by one

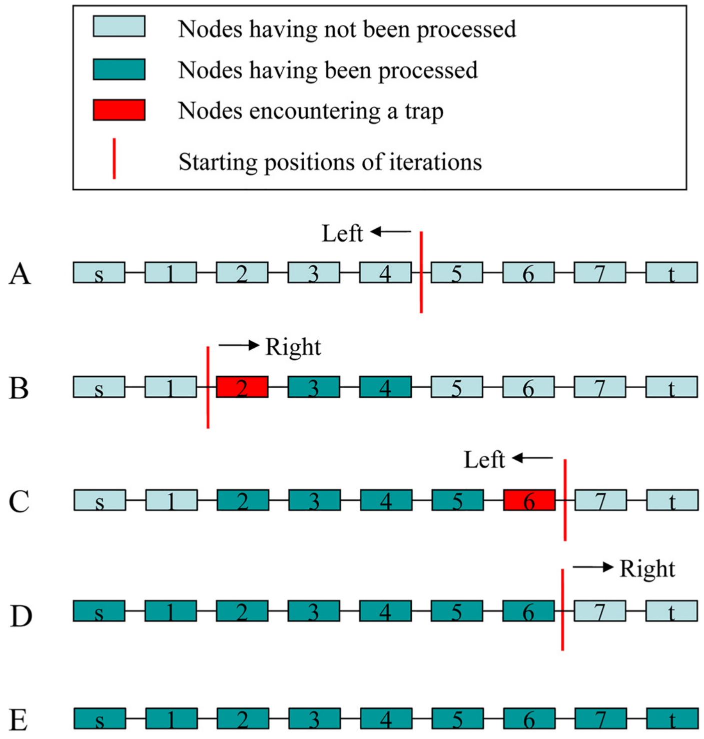
Nodes having not been processed
Nodes having been processed
Nodes encountering a trap
Starting positions of iterations
A
Left
s 1 2 3 4 5 6 7 t
B
Right
s 1 2 3 4 5 6 7 t
C
Left
s 1 2 3 4 5 6 7 t
D
Right
s 1 2 3 4 5 6 7 t
E
s 1 2 3 4 5 6 7 t

Fig 7. An example which shows the iterations of BinPacker in the packing process. (A) The first iteration of BinPacker starts from node 4 to the left. (B) When BinPacker processes nodes one by one to the left, it encounters a trap node 2, and then enters the next iteration starting from node 2 to the right. (C) When BinPacker processes nodes to the right, it encounters another trap node 6 and then enters the next iteration starting from node 6 towards left. (D) In this iteration, BinPacker reaches a terminal *s*, and then jumps back to the starting node 6 and processes the remaining nodes one by one to the right. (E) BinPacker reaches another terminal *t* and terminates its iterations.

doi:10.1371/journal.pcbi.1004772.g007

until encountering a new trap node or reaching the sink node t. Repeat the procedure until all nodes are processed (see Fig 7). The forth and back iteration must terminate within a few times (no more than $|V|$ times, where V is the node set of the splicing graph) due to the fact that the nodes previously processed will never be trapped again (see Theorem 2 in S1 Text).

Step 3: Bin packing by 0–1 quadratic programming. Suppose that BinPacker is processing the node n_L towards left and we have m edges (bins) coming to n_L from which the bins (edges) containing n items go out.

If $m \leq n$, the bin packing can be solved by the following 0–1 quadratic programming:

$$\min f = \sum_{i=1}^{m}\left(c_i - \sum_{j=1}^{n} w_j x_{ij}\right)^2$$
$$s.t. \begin{cases} \sum_{i=1}^{m} x_{ij} = 1 & \forall j = 1, \ldots, n \\ \sum_{j=1}^{n} x_{ij} \geq 1 & \forall i = 1, \ldots, m \\ x_{ij} \in \{0, 1\} \end{cases} \qquad (1)$$

where c_i represents the capacity of bin i, w_j the size of item j, and x_{ij} is a binary variable with $x_{ij} = 1$ if item j is packed into bin i, 0 otherwise. The first constraint ensures that one item goes into one and only one bin, and the second one guarantees that each bin receives at least one item.

Otherwise, we have $m>n$, and get trapped at the node n_L. Then the bin packing can be solved by the following quadratic programming:

$$\min f = \sum_{i=1}^{\mathrm{k}} (c_i - \sum_{j=1}^{m} w_j x_{ij})^2$$

$$s.t. \quad \begin{cases} \sum_{i=1}^{\mathrm{k}} x_{ij} = 1 \quad \forall j = 1, \ldots, m \\ \sum_{j=1}^{m} x_{ij} \geq n_i \quad \forall i = 1, \ldots, \mathrm{k} \\ x_{ij} \in \{0, 1\} \end{cases} \tag{2}$$

where k represents the number of edges going out of n_L, n_i the number of items packed into bin (edge) i ($1 \leq i \leq k$). This constraint ensures that each bin previously packed will get at least the same number of items as in the last iteration.

Step 4: Transformation into 0–1 ILP. The 0–1 quadratic programming can be equivalently transformed into a 0–1 linear programming (see Methods and Theorem 3 in S1 Text for details). To do so, we simply introduce a new variable x_{ijik} for each quadratic term $x_{ij} \cdot x_{ik}$ (or $x_{ik} \cdot x_{ij}$) in the objective function along with the constraints:

$$\begin{cases} x_{ijik} \leq x_{ij} \quad \forall i = 1, \ldots, m \quad 1 \leq j \leq k \leq n \\ x_{ijik} \leq x_{ik} \quad \forall i = 1, \ldots, m \quad 1 \leq j \leq k \leq n \\ x_{ij} + x_{ik} - 1 \leq x_{ijik} \quad \forall i = 1, \ldots, m \quad 1 \leq j \leq k \leq n \\ x_{ijik} \in \{0, 1\} \quad \forall i = 1, \ldots, m \quad 1 \leq j \leq k \leq n \end{cases}$$

4. Recovery of an optimal set of full-length transcripts

All the 0–1 ILPs are optimally solved by GLPK-4.40. The GLPK (GNU Linear Programming Kit) package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is a set of routines written in ANSIC and organized in the form of a callable library. Since each programming is modeled from one node of a splicing graph, the number of variables of the 0–1 ILP is $|I_{in}| \cdot |I_{out}| \cdot (|I_{out}|+3)/2$ or $|I_{out}| \cdot |I_{in}| \cdot (|I_{in}|+3)/2$. In most cases, $|I_{out}|<3$ and $|I_{in}|<3$, so the number of variables of the 0–1 ILP is almost always less than 27 and it can be optimally solved by GLPK extremely fast. And even in many cases, $|I_{out}| = 1$ or $|I_{in}| = 1$, in which cases, items can be directly packed into corresponding bins without using GLPK.

The solution $\{x_{ij}\}$ tells us that item j is in bin i if and only if $x_{ij} = 1$. All the bins (edges) containing the same item induce an s-t path in the splicing graph of a gene which may correspond to a transcript of the gene. Finally, BinPacker outputs all the transcripts induced by individual items in the splicing graph of the gene.

Supporting Information

S1 Text. Supplementary material of BinPacker.
(PDF)

Author Contributions

Conceived and designed the experiments: GL. Performed the experiments: JL. Analyzed the data: JL TY BL. Contributed reagents/materials/analysis tools: JL ZC. Wrote the paper: JL GL XH. Designed the software used in analysis: JL TY RM PC. Oversaw the project: GL XH.

References

1. Sharon D, Tilgner H, Grubert F, Snyder M (2013) A single-molecule long-read survey of the human transcriptome. Nat Biotechnol 31: 1009–1014. doi: 10.1038/nbt.2705 PMID: 24108091
2. Bao E, Jiang T, Girke T (2013) BRANCH: boosting RNA-Seq assemblies with partial or related genomic sequences. Bioinformatics 29: 1250–1259. doi: 10.1093/bioinformatics/btt127 PMID: 23493323
3. Metzker ML (2009) Sequencing technologies—the next generation. Nature Reviews Genetics 11: 31–46. doi: 10.1038/nrg2626 PMID: 19997069
4. Matlin AJ, Clark F, Smith CW (2005) Understanding alternative splicing: towards a cellular code. Nat Rev Mol Cell Biol 6: 386–398. PMID: 15956978
5. Black DL (2003) Mechanisms of alternative pre-messenger RNA splicing. Annu Rev Biochem 72: 291–336. PMID: 12626338
6. Wang ET, Sandberg R, Luo S, Khrebtukova I, Zhang L, et al. (2008) Alternative isoform regulation in human tissue transcriptomes. Nature 456: 470–476. doi: 10.1038/nature07509 PMID: 18978772
7. Martin JA, Wang Z (2011) Next-generation transcriptome assembly. Nature Reviews Genetics 12: 671–682. doi: 10.1038/nrg3068 PMID: 21897427
8. Grabherr MG, Haas BJ, Yassour M, Levin JZ, Thompson DA, et al. (2011) Full-length transcriptome assembly from RNA-Seq data without a reference genome. Nature biotechnology 29: 644–652. doi: 10.1038/nbt.1883 PMID: 21572440
9. Haas BJ, Zody MC (2010) Advancing RNA-Seq analysis. Nature Biotechnology 28: 421–423. doi: 10.1038/nbt0510-421 PMID: 20458303
10. Trapnell C, Williams BA, Pertea G, Mortazavi A, Kwan G, et al. (2010) Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. Nature biotechnology 28: 511–515. doi: 10.1038/nbt.1621 PMID: 20436464
11. Guttman M, Garber M, Levin JZ, Donaghey J, Robinson J, et al. (2010) Ab initio reconstruction of cell type-specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincRNAs. Nature biotechnology 28: 503–510. doi: 10.1038/nbt.1633 PMID: 20436462
12. Pertea M, Pertea GM, Antonescu CM, Chang TC, Mendell JT, et al. (2015) StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. Nature Biotechnology 33: 290-+. doi: 10.1038/nbt.3122 PMID: 25690850
13. Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJM, et al. (2009) ABySS: A parallel assembler for short read sequence data. Genome Research 19: 1117–1123. doi: 10.1101/gr.089532.108 PMID: 19251739
14. Xie YL, Wu GX, Tang JB, Luo RB, Patterson J, et al. (2014) SOAPdenovo-Trans: de novo transcriptome assembly with short RNA-Seq reads. Bioinformatics 30: 1660–1666. doi: 10.1093/bioinformatics/btu077 PMID: 24532719
15. Schulz MH, Zerbino DR, Vingron M, Birney E (2012) Oases: robust de novo RNA-seq assembly across the dynamic range of expression levels. Bioinformatics 28: 1086–1092. doi: 10.1093/bioinformatics/bts094 PMID: 22368243
16. Peng Y, Leung HC, Yiu SM, Lv MJ, Zhu XG, et al. (2013) IDBA-tran: a more robust de novo de Bruijn graph assembler for transcriptomes with uneven expression levels. Bioinformatics 29: i326–334. doi: 10.1093/bioinformatics/btt219 PMID: 23813001
17. Chang Z, Li GJ, Liu JT, Zhang Y, Ashby C, et al. (2015) Bridger: a new framework for de novo transcriptome assembly using RNA-seq data. Genome Biology 16.
18. Robertson G, Schein J, Chiu R, Corbett R, Field M, et al. (2010) De novo assembly and analysis of RNA-seq data. Nature Methods 7: 909–U962. doi: 10.1038/nmeth.1517 PMID: 20935650

19. Kent WJ (2002) BLAT—the BLAST-like alignment tool. Genome research 12: 656–664. PMID: 11932250

20. Rosenbloom KR, Armstrong J, Barber GP, Casper J, Clawson H, et al. (2015) The UCSC Genome Browser database: 2015 update. Nucleic Acids Research 43: D670–D681. doi: 10.1093/nar/gku1177 PMID: 25428374

21. Cunningham F, Amode MR, Barrell D, Beal K, Billis K, et al. (2015) Ensembl 2015. Nucleic Acids Research 43: D662–D669. doi: 10.1093/nar/gku1010 PMID: 25352552

22. Griebel T, Zacher B, Ribeca P, Raineri E, Lacroix V, et al. (2012) Modelling and simulating generic RNA-Seq experiments with the flux simulator. Nucleic Acids Research 40: 10073–10083. doi: 10.1093/nar/gks666 PMID: 22962361

23. Zhao QY, Wang Y, Kong YM, Luo D, Li X, et al. (2011) Optimizing de novo transcriptome assembly from short-read RNA-Seq data: a comparative study. Bmc Bioinformatics 12.

24. Heber S, Alekseyev M, Sze SH, Tang H, Pevzner PA (2002) Splicing graphs and EST assembly problem. Bioinformatics 18 Suppl 1: S181–188. PMID: 12169546

25. Kahn AB (1962) Topological sorting of large networks. Communications of the ACM 5: 558–562.