Yu et al. Genome Biology (2018) 19:167
https://doi.org/10.1186/s13059-018-1535-9

Genome Biology

METHOD Open Access

SeqOthello: querying RNA-seq experiments at scale

Ye Yu[1], Jinpeng Liu[1], Xinan Liu[1], Yi Zhang[1], Eamonn Magner[1], Erik Lehnert[2], Chen Qian[3] and Jinze Liu[1*]

Abstract

We present SeqOthello, an ultra-fast and memory-efficient indexing structure to support arbitrary sequence query against large collections of RNA-seq experiments. It takes SeqOthello only 5 min and 19.1 GB memory to conduct a global survey of 11,658 fusion events against 10,113 TCGA Pan-Cancer RNA-seq datasets. The query recovers 92.7% of tier-1 fusions curated by TCGA Fusion Gene Database and reveals 270 novel occurrences, all of which are present as tumor-specific. By providing a reference-free, alignment-free, and parameter-free sequence search system, SeqOthello will enable large-scale integrative studies using sequence-level data, an undertaking not previously practicable for many individual labs.

Keywords: RNA-seq, TCGA, Gene fusion, Pan-cancer, Query, Compression, Othello, SeqOthello

Background

Advances in the study of functional genomics over the past decade have produced a vast resource of RNA-seq datasets. As of December 2017, over 12 petabytes of RNA-seq data were deposited in the Sequence Read Archive (SRA) [1]. Sequencing consortiums such as The Cancer Genome Atlas (TCGA) [2] and the International Cancer Genomics Consortium (ICGC) [3] have sequenced tens of thousands of tumor transcriptomes from diverse cancer populations. Although these datasets have collectively redefined the landscape of cancer transcriptomes, additional clinically relevant features remain to be discovered. However, data reanalysis to identify these features requires extensive computational resources and bioinformatics support, making it exclusive to a few labs. The development of SeqOthello will enable labs with limited resources to learn from sequence-level data by supporting fast and memory-efficient query over large-scale RNA-seq datasets.

To date, sequence search options are limited. Most sequencing databases support metadata searches [1, 3, 4], which permit selection of experiments by tissue type, organism, experimental condition or sequencing protocol. From this refined list, experiments can be downloaded and analyzed individually [5]. SRA-BLAST [6] is able to search only a limited set of sequencing experiments. Finally, the bioinformatics community has lately established databases storing ready-to-analyze results in areas such as gene or transcript expression [4, 7–9]. However, these databases are subject to frequent updates as bioinformatics algorithms improve and reference genomes are refined, nor can they support the query of novel sequences that are absent from existing annotation or undetectable by current bioinformatics tools.

Recently, Sequence Bloom Tree (SBT) [10] and its descendants [11, 12] were developed to query RNA-seq experiments for expressed transcripts, pioneering the field of large-scale sequence search in RNA-seq. SBT is designed as an experiment filter that returns the subset of experiments containing at least θ percent of k-mers from the query sequence. Built upon bloom filters [13, 14], SBT-based algorithms are generally memory efficient for small queries. Unfortunately, tuning the input parameter θ is time-consuming and produces inconsistent results for a single query, thereby hampering interpretability. Furthermore, extracting sequence-level information from the filtered experiments requires downloading and reanalyzing the raw sequencing datasets and thus does not sidestep traditional RNA-seq processing. Very recently, Mantis [15] (by Pandey et al.) used counting quotient filter to further improve the speed in sequence

* Correspondence: liuj@cs.uky.edu
[1]Department of Computer Science, University of Kentucky, 301 Rose St, Lexington, KY 40508, USA
Full list of author information is available at the end of the article

search. There is also growing interest in methods for indexing large collections of genomic sequencing reads from different individuals. Bloom Filter Trie (BFT) [16] was developed to store and compress a set of colored *k*-mers from a Pan-Genome of hundreds of samples. Additionally, the Burrows–Wheeler transform (BWT) and FM index have been employed to build indexes on raw sequencing reads with applications in compressing 2705 whole genome sequencing samples from the 1000 Genomes Project [17, 18]. Though retaining full-text information, these data structures are often associated with high memory cost and slow query speed as the entire index must be loaded to memory prior to query.

Here we present SeqOthello, a novel indexing structure that supports query of an arbitrary sequence against large collections of RNA-seq experiments. Large-batch query with SeqOthello is orders of magnitude faster than with SSBT, the improved version of SBT. A SeqOthello query may return near-exact *k*-mer information in individual experiments or *k*-mer hit ratios (i.e., the fraction of *k*-mer hits in a query). We illustrate the utility and efficiency of SeqOthello by conducting a global survey of known gene fusions against 10,113 TCGA RNA-seq datasets. The survey confirms roughly 93% of known fusion events and reveals 270 novel occurrences, all of which are tumor-specific. Index construction on over 10,000 *k*-mer files, representing RNA-seq datasets extracted from TCGA, required less than nine CPU hours on a computer with 32 GB memory. The entire survey only took under 5 min, which, to our knowledge, is a scale unachieved by previous methods.

Results

SeqOthello data structure

A sequencing experiment can be represented by a collection of *k*-mers, or length *k* subsequences of the original reads. *k*-mers are fundamental components of de Bruijn graphs and are essential for de novo transcriptome assembly [19–21]. A *database* of sequencing experiments can therefore be represented as a collection of *occurrence maps* of individual *k*-mers. The occurrence map of a *k*-mer is defined as its presence or absence across all experiments indexed in the database. The challenge is to efficiently store and query this information in scenarios with billions of *k*-mers across tens of thousands of experiments. We leverage novel algorithms in data compression and *k*-mer indexing to surmount this obstacle.

The prevalence of each *k*-mer varies dramatically, with plots of *k*-mer frequency often exhibiting a *U*- or *L*-shaped distribution (Additional file 1: Figure S1). *k*-mers located at the extremes of the spectrum tend to originate from experiment-specific transcripts or common transcripts that manifest in nearly all experiments. By contrast, *k*-mers near the center of the distribution may be tissue- or organism-specific. The prevalence of a *k*-mer directly determines the information content [22, 23] or the number of bits required to store its occurrence map. To this end, SeqOthello employs an information-content-aware data-compression scheme: an ensemble of compression techniques tailored to store the occurrence maps of *k*-mers from each region of the occurrence distribution without hampering query efficiency (Fig. 1a and Methods). SeqOthello relies on a novel, hierarchical indexing structure to facilitate fast retrieval of *k*-mer occurrence maps (Fig. 1a). The mappings between levels are supported by the *Othello* data structure [24, 25] (Methods), a minimal perfect hashing classifier that provides key-to-value searching in constant time. An Othello is significantly more compact than a traditional hash table as it does not store keys. But an Othello constructed on billions of *k*-mers still demands too much memory to be practical for use with standard computers. The hierarchical structure employed by SeqOthello overcomes this challenge using a divide-and-conquer approach. Specifically, *k*-mer occurrence maps are split into buckets according to their encoded lengths, with the assignment of each *k*-mer to its bucket determined by the root Othello. Within each bucket, the mapping between a *k*-mer and the location of its occurrence map is again stored in an Othello. SeqOthello significantly increases the volume of indexed *k*-mers within limited memory space and is inherently parallelizable.

Querying a SeqOthello first requires decomposing the query sequence into its constituent *k*-mers. The root Othello node identifies the occurrence bucket for each *k*-mer, following which each bucket Othello node retrieves the desired occurrence map. Per *k*-mer, this process requires exactly two Othello queries and is thus executed in constant time. The full set of occurrence maps is then synthesized to generate a *k*-mer hit map of the query for each experiment, where a hit means a *k*-mer is present in an experiment. Each *k*-mer hit map can be summarized into the number of hits or a hit ratio, the fraction of hits out of the total *k*-mers in the query (Fig. 1b).

SeqOthello outperforms state-of-the-art algorithms

We compare SeqOthello to each of three state-of-the-art methods for querying large-scale RNA-seq datasets: SBT [10], SSBT [11], and SBT-AS [12]. The evaluation was benchmarked on 2652 RNA-seq experiments of human blood, breast, and brain tissues from the SRA (Additional file 2). We use Jellyfish [26] to convert raw sequence data into *k*-mer files at a rate of 1.85 min per file. Taking these files as input, SeqOthello requires 1.93 h and a

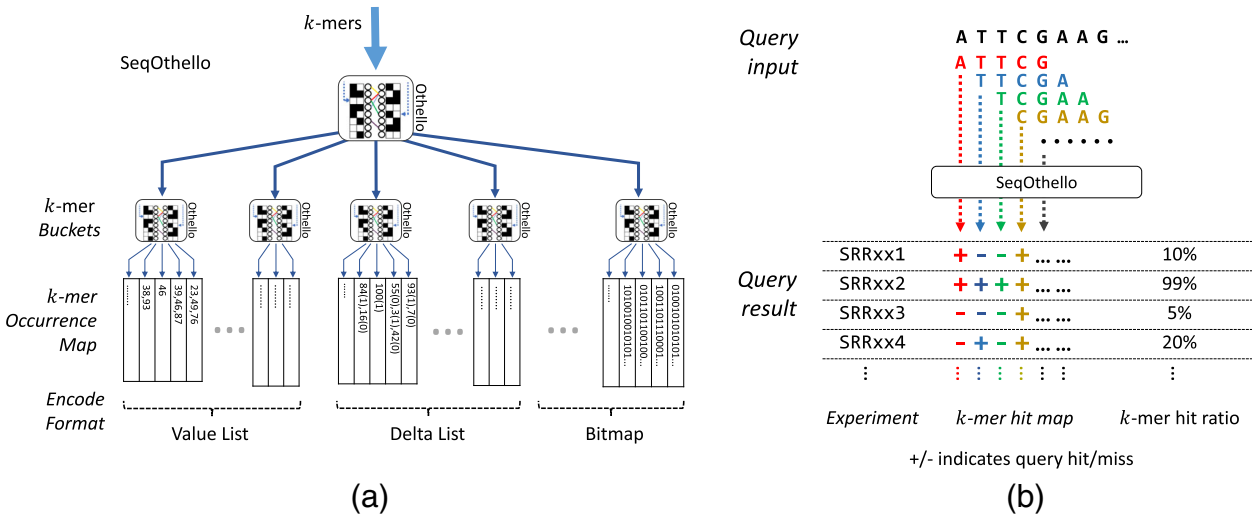

Fig. 1 Overview of SeqOthello structure and query procedure. **a** An illustration of the SeqOthello indexing structure to support scalable *k*-mer searching in large-scale sequencing experiments. The bottom level of SeqOthello stores the occurrence maps of individual *k*-mers, encoded in three different formats and divided into disjoint buckets. The mapping between a *k*-mer and its occurrence map is achieved by a hierarchy of Othello structures in which the root Othello maps a *k*-mer to its bucket and the Othello in each bucket maps a *k*-mer to its occurrence map. **b** An example illustrating SeqOthello's sequence query process and output. A sequence query is decomposed into its constituent *k*-mers. The query result can be either a *k*-mer hit map, recording each *k*-mer's presence/absence along the query sequence, or *k*-mer hit ratios (i.e., the fraction of query *k*-mers present in each experiment)

maximum of 14.1 GB memory to construct the index, 10 times faster than SBT and SSBT. At 20.8 GB, the SeqOthello index is 30% smaller than the most-compact SBT-based index, SSBT, and achieves a 700:1 compression ratio relative to the original database (Additional file 3: Table S1).

SeqOthello queries 198,093 transcripts from Gencode Release 25 [27] for *k*-mer hits in all 2652 experiments in 35.7 min using 15.2 GB memory. With four threads, the running time drops to 13.4 min. SBT-based queries only return the set of experiments whose *k*-mer hit ratio is greater than a user-defined threshold, denoted by θ. Even with a very high *k*-mer hit ratio ($\theta = 0.9$), SBT-AS and SBT require 575 and 4160 min to complete, respectively, with higher memory cost than SeqOthello (Fig. 2). While SSBT is extremely memory frugal, it is at the expense of

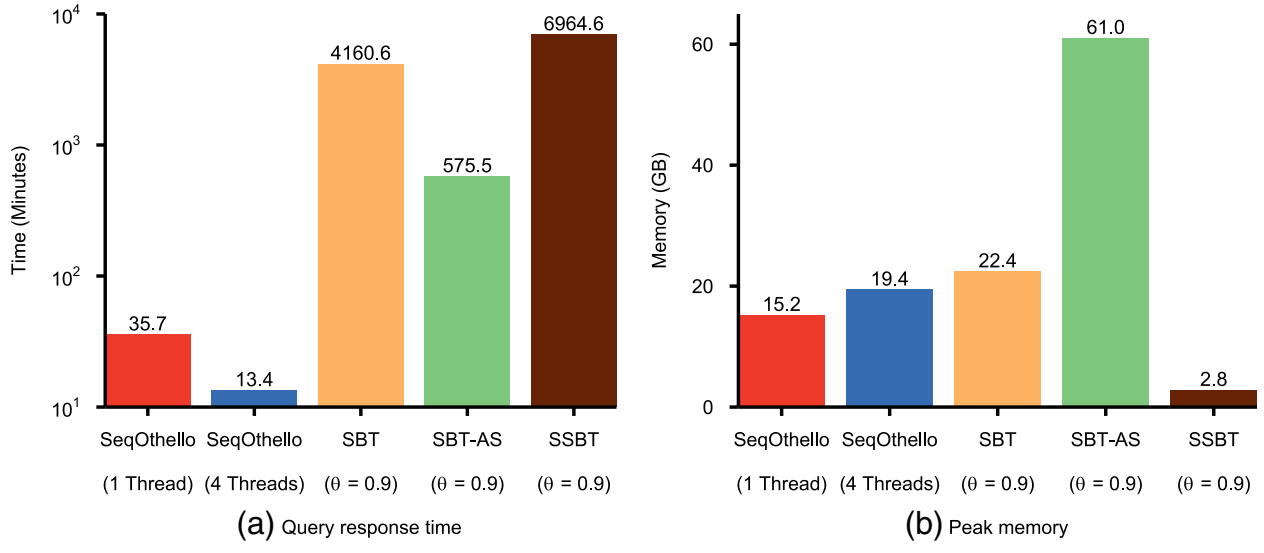

Fig. 2 Comparing query performance for SeqOthello and three SBT-based algorithms: SBT [10], SSBT [11], and SBT-AS [12]. Performance is benchmarked on 2652 human RNA-seq experiments. The query consists of 198,093 human transcripts in Gencode Release 25. **a** Query response time. **b** Peak memory

much slower speed, two orders of magnitude slower than SeqOthello (Fig. 2).

The significance of experiments extracted by SBT using a single threshold θ is difficult to assess. To avoid generating misleading conclusions, multiple queries with different θ may be attempted to determine an approximate distribution, affording an overall query time several times larger than we report. Querying a small batch of 1000 transcripts with settings of $\theta = 0.7$, $\theta = 0.8$, and $\theta = 0.9$ required 40 min to execute with SBT-AS, 190 min with SBT, and 241 min with SSBT (Additional file 4: Table S2). In contrast, SeqOthello requires only 4.6 min to query the same set of transcripts and generates exact hit ratios for each transcript in each indexed experiment.

SeqOthello also accommodates online features for small-batch queries. Online queries preload the entire index into memory prior to querying and can be executed in approximately 0.09 s per transcript (Additional file 4: Table S2). Our method's advantageous speed supports on-demand queries from multiple users in a client-server setting. Other methods do not have online options at present.

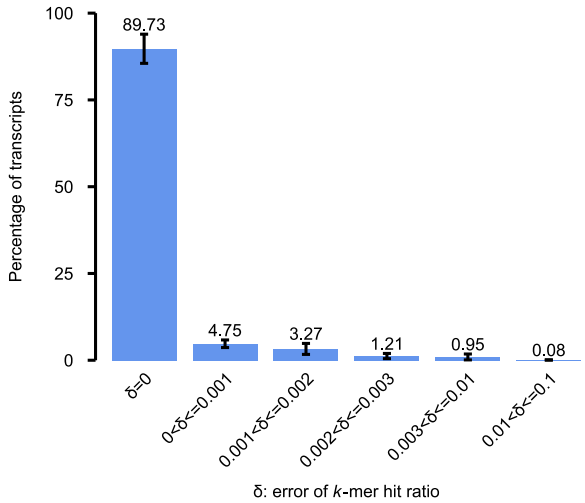

Fig. 3 The distribution of error rate in *k*-mer hit ratios returned by SeqOthello. A randomly selected set of 150 experiments are extracted from SeqOthello's result by querying all human transcripts on 2652 human experiments. The error (δ) of a transcript query over an experiment is calculated as the difference between the transcript's *k*-mer hit ratio returned by SeqOthello and the *k*-mer hit ratio obtained by mapping raw *k*-mers using the same RNA-seq experiments to the transcript sequences. Each bar shows the percentage of transcripts with δ falling in a specific range. The error bar shows the standard deviation of such percentage measured on 150 experiments

SeqOthello achieves near-exact *k*-mer query

SeqOthello always returns the correct occurrence map when querying *k*-mers that are present in at least one experiment sample. However, for queries involving alien *k*-mers that are not present in any of the original experiments, SeqOthello may return false-positive occurrences. (Methods, Section 4). To assess the accuracy in general *k*-mer search, we queried 120,044,842 *k*-mers present in human transcriptome Gencode Release 25 against the SeqOthello index constructed for the aforementioned 2652 experiments. We randomly selected 150 experiments and calculated the false-positive rate of *k*-mer queries in each experiment. The false-positive rate is defined as the fraction of *k*-mers absent from the raw *k*-mer file that SeqOthello classifies as present among all queried *k*-mers. The Venn diagram (Additional file 5: Figure S2) shows overlap among three sets of *k*-mers. For *k*-mers that are not present in any of the indexed experiments, SeqOthello yields an extremely low rate of false positives: across 150 randomly chosen experiments, the average false-positive rate was 0.015% with standard deviation of 0.071%.

To further evaluate the effect of false positives on transcript queries, we mapped the raw *k*-mers of each experiment to transcript sequences, calculating the true *k*-mer hit ratio for each transcript. We then compared the *k*-mer hit ratios generated by SeqOthello to the ground truth. Roughly 89.7% of transcripts afforded *k*-mer hit ratios equal to the true value, with an additional 9.3% exhibiting an error rate up to 0.003 (Fig. 3). These results demonstrate that SeqOthello achieves near-exact query of *k*-mers and *k*-mer hit ratios. Additionally, as consecutive *k*-mers in a sequence are highly redundant, even a single base mismatch to the query sequence will be evidenced by the absence of multiple (i.e., *k*) *k*-mers, rendering an extremely low likelihood of false-positive match due to alien *k*-mers (Methods). Although *k*-mer information is implicitly stored in bloom filters employed in SBT-based algorithms, efficient implementation of *k*-mer retrieval by these algorithms is not yet available.

SeqOthello enables efficient query against TCGA Pan-Cancer RNA-Seq experiments

The Cancer Genome Atlas (TCGA) [2] contains transcriptome profiles of 10,113 tumor samples obtained from 9215 cancer patients. The database allows researchers to detect and characterize novel transcriptomic alterations across 29 different cancer types in the GDC Legacy Archive [2]. We have constructed a SeqOthello index, storing the occurrences of 1.47 billion 21-mers across all tumor samples (Additional file 6). The preparation of *k*-mers averaged 4 min per sample while the construction of SeqOthello on all samples required less than 9 h. The index occupies only 76.6 GB of space, thus is portable for querying at different locations.

We use the SeqOthello index to conduct a survey of all gene-fusion events curated by TCGA Fusion Gene Database as of December 2017 [28]. The database contains 11,658 documented unique tier-1 fusion events from TCGA detected by PRADA [29]. This represents 10,994 gene fusion pairs as multiple junctions might exist for one fusion pair. For each fusion junction, we construct a fusion sequence that will be used to query SeqOthello for its presence. The sequence consists of 20 bases from the donor exon and 20 bases from the acceptor exon, thereby guaranteeing that any 21-mer from the sequence will span the fusion junction (Additional file 7: Figure S3).

A SeqOthello query of a fusion sequence returns the number of *k*-mer hits in each sample. A simple fusion-calling method may take an SBT-like approach, requiring a minimum fraction of *k*-mer hits, θ, to call the presence in each sample. However, this technique yields lackluster sensitivity and specificity. Lowering θ permits fusion detection with fewer spanning reads, but may increase false-positive calls if the fusion junction sequence contains repetitive *k*-mers that are abundant in many samples. Instead of using a fixed threshold for all fusion calls, we develop a noise-aware approach. The approach first evaluates the background noise generated from repetitive *k*-mers present in a large fraction of samples. This is quantified by leveraging the distribution of *k*-mer hits across TCGA tumor samples queried through SeqOthello. Two examples with different levels of repetitive *k*-mers are shown in Fig. 4a, b. Assume true fusion occurs in less than 2% of samples, as TMPRSS2-ERG achieves the highest occurrence to date at 0.953% of all TCGA tumor samples [28] (14.657% occurrence rate in prostate tumor samples). For each fusion, we estimate the level of background noise, δ, as the number of *k*-mer hits at the 98th percentile of the samples in the distribution of *k*-mer hits. We require an additional number of *k*-mers, μ, beyond the background noise as evidence of expression to conclude the fusion occurrence in a sample. We compared the noise-aware approach with the θ-based SBT-like approach in recovering known fusion occurrences and in detecting unknown fusion occurrences. As shown in Fig. 4c, the noise-aware approach recovers more known fusions than the SBT-like approach without generating too many putative fusions that are likely to be false. Fusion occurrences called at $\mu = 7$ is used for further analysis as it renders the best sensitivity while being most conservative in generating candidates of novel occurrences. We then compared the distributions of actual *k*-mer hits of known fusion occurrences and novel occurrences in all the called fusion occurrences. The consistency between known and novel occurrences across the entire spectrum of *k*-mer hits further supports the validity of the noise-aware approach (Fig. 4d). Please note that the background noise δ should be adjusted according to the expected signal to noise ratio in individual applications.

Under this method, we detect 92.7% of tier-1 fusion occurrences in TCGA Fusion Gene Database [28] with at least 10 spanning reads reported by PRADA. Additionally, we identify 270 novel occurrences of fusion events across 17 tumor subtypes that are not identified by PRADA. We selected two fusion pairs with occurrences most inconsistent with current curation for further validation: FGFR3-TACC3 in GBM samples (5 novel, 3 undetected) and ESR1-C6orf97 in BRCA samples (2 novel, 5 undetected). We confirmed all 7 novel occurrences by identifying at least 10 fusion spanning reads supporting each. For all undetected fusions, insufficient spanning reads were confirmed, consistent with low read support cited in the database (Additional file 6).

Figure 5 depicts the 10 novel, recurring fusions with greatest number of occurrences suggested by SeqOthello. Several have doubled or even tripled the original recurring rates. Interestingly, all novel occurrences agree with the original fusion cancer-type classifications, rendering the chance of random occurrence negligible. This result corroborates their cancer specificity and supports the high precision of SeqOthello's query results. One example of this consistency is TMPRSS2-ERG, a clinical marker for prostate cancer. SeqOthello extracted 122 pre-identified occurrences of TMPRSS2-ERG and 142 novel occurrences, all from prostate cancer samples. The complete information of all detected fusion occurrences is listed in Additional file 6.

Discussion

SeqOthello is a novel algorithm capable of indexing large-scale RNA-seq experiments that supports online sequence query. We constructed a SeqOthello index on the TCGA Pan-Cancer RNA-seq datasets, the latter totaling 54 TB in compressed fastq format. The SeqOthello index uses only a 76.6-GB disk space, achieving a compression ratio of 700:1. Querying the index to assess the prevalence of 11,658 documented fusion events requires only 5 min on a standard desktop computer with 32 GB memory. The index circumvents the need to reanalyze large volumes of sequencing datasets in searching for novel transcripts, which is computationally prohibitive. For example, it is estimated to take 54 days of computational time when repurposing the fastest RNA-seq aligner to achieve targeted fusion search (Methods, Section 6). Thus, SeqOthello provides an

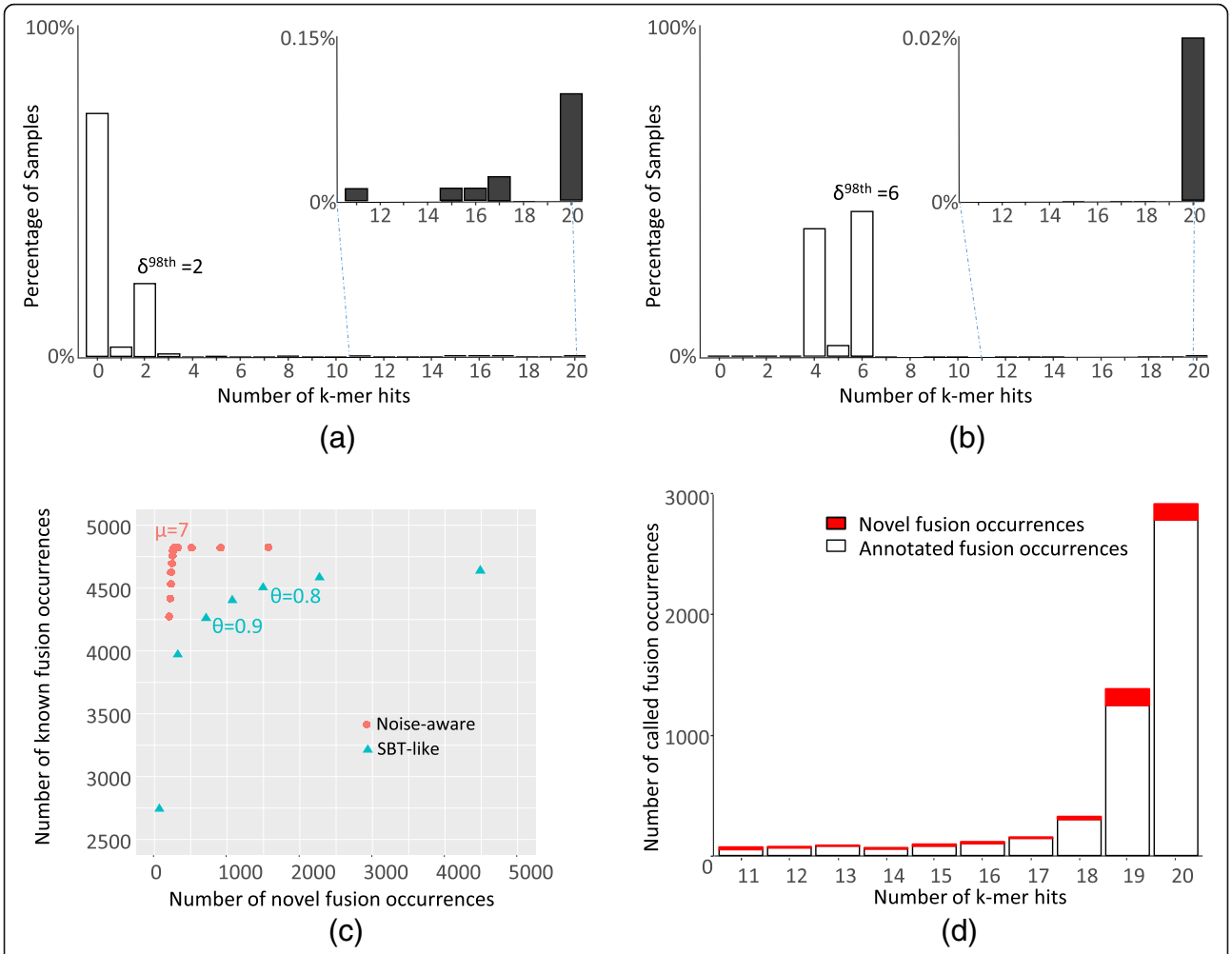

Fig. 4 An illustration of fusion calling criteria using SeqOthello's query results against TCGA RNA-seq data. **a**, **b** Examples of *k*-mer hit distribution as a result of fusion junction sequence query using SeqOthello. The presence of a small set of *k*-mers in large fraction of samples indicates background noise as a result of these *k*-mers being repetitive. For each fusion, we use δ^{98th}, the *k*-mer hit at 98th percentile as an estimation of background noise. **a** Histogram of *k*-mer hits querying junction sequence spanning chr21:42880008-chr21:39956869 connecting gene pair TMPRSS2-ERG. The background noise is estimated at $\delta^{98th} = 2$. **b** Histogram of *k*-mer hits querying junction sequence spanning chr5:134688636-chr5:179991489 connecting gene pair H2AFY-CNOT6. The background noise is estimated at $\delta^{98th} = 6$. **c** The comparison of performance in recovering database-known fusion occurrences and detecting novel occurrences between noise-aware approach and SBT-like approach using θ-based containment query. Here μ is the minimum number of *k*-mer hits required beyond the fusion-specific noise level used in the noise-aware approach. The change in μ between two adjacent points is 1; θ is the minimum fraction of *k*-mer hits required to call the presence of a query as used in SBT containment query. The change in θ between two adjacent points is 0.05. **d** The distribution of the actual *k*-mer hits of all fusion occurrences called with the noise-aware approach

efficient and viable solution for on-demand sequence query against a large-scale sequencing database, overcoming the barrier in data access by the broader research community.

SeqOthello can be parameterized to report either the number of *k*-mer hits of a query sequence or the presence/absence information of each constituent *k*-mer of the query in individual samples. The utility of SeqOthello's query result is demonstrated by its application to gene fusion survey, accurately determining the tumor-specificity of individual fusion events without requiring downloading and reanalysis of raw sequencing data.

The simple query supported by SeqOthello is powerful, with myriad applications yet to be defined. One can use SeqOthello to assess the prevalence of clinically important features in different patient populations or to compare across different patient cohorts. Beyond transcripts, one can use SeqOthello to identify expressed regions by querying entire reference genomes. SeqOthello can be potentially leveraged on any form of next-generation sequencing data that can be translated to a *k*-mer occurrence matrix. We leave the definitions and demonstrations of these applications for future work.

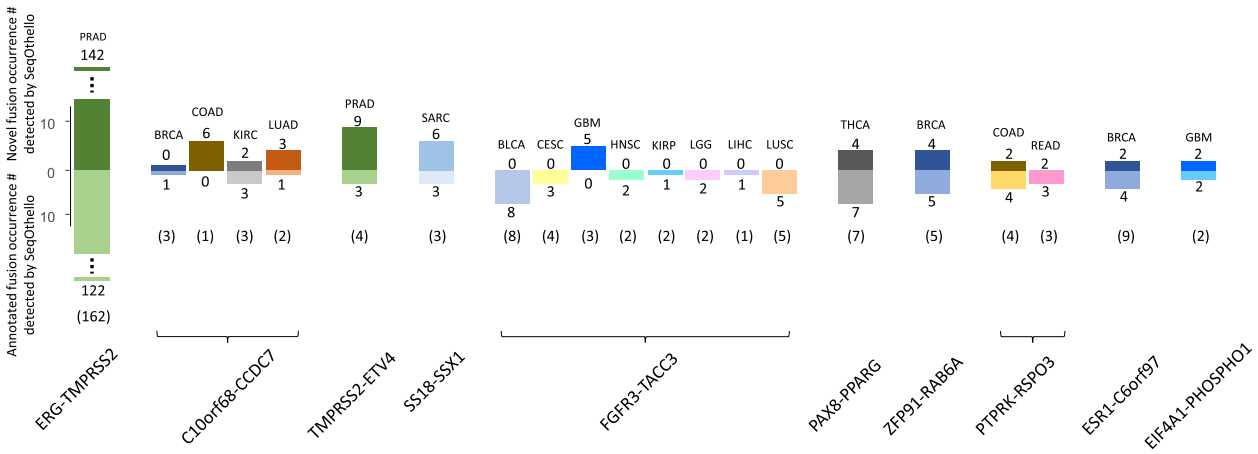

Fig. 5 Top ten most recurring gene fusion events queried through SeqOthello indexing 10113 TCGA RNA-seq experiments across 29 tumor types.. Bar plots show occurrence number of top ten recurrent gene fusions detected by SeqOthello over different tumor types. Occurrences of each fusion on each tumor type are classified into novel occurrences (not reported in TCGA Gene Fusion Database) and annotated occurrences (already curated by TCGA Gene Fusion Database)

Conclusion

SeqOthello supports parameter-free, reference-free and annotation-free sequence query against large collection of sequencing experiments. Its unbiased nature supports large-scale integrative and comparative studies, while its ultra-fast performance and undemanding system requirements render it appropriate for a wide variety of research investigators. SeqOthello will enable novel discoveries that would be otherwise unrealizable for individual research labs.

Methods

Section 1. The Othello data structure

The mapping of *k*-mers in either level of SeqOthello is maintained by a data structure named *Othello*. Othello belongs to the class of minimal perfect hashing (MPHF) algorithms [30]. However, conventional MPHF supports one-to-one mapping between a predefined set of keys to a set of integers, so that each key corresponds to a unique integer. Unlike MPHF, Othello conducts many-to-1 mapping where more than one key can be mapped to the same integer with the condition that one key can be only mapped to one integer. Thus, Othello naturally implements a hashing classifier that efficiently maps keys (*k*-mers) to appropriate categories. To date, the Othello algorithm has demonstrated great scalability of both memory and querying speed in various applications [24, 25, 31].

An Othello $\boldsymbol{O}(S,V)$ maps a predefined set of *k*-mers S to a list of categories represented as integers, denoted by $V = \{1, 2, \cdots, v\}$. Let $T: S \rightarrow V$ be the function that maps *k*-mers in S to classes in V, where $T(s)$ indicates the category of a *k*-mer $s \in S$. Each category in V is represented by an l-bit integer, where $l = \lceil \log_2(v+1) \rceil$.

In essence, an Othello $\boldsymbol{O}(S,V)$ maintains a query function $\tau: U \rightarrow C$ mapping the set of all possible *k*-mers, U, to the set of all l-bit integers, $C = \{0, 1, \cdots, 2^l - 1\}$. Thus $S \subset U$ and $V \subset C$. Furthermore, τ is a superset of T: That is, for any $s \in S$, $\tau(s) = T(s)$; for any $s' \in U - S$, $\tau(s)$ is a deterministic l-bit integer. A *k*-mer s' is called *alien* if and only if $s' \in U - S$, such that $s' \notin S$ and the mapping for s' is not specified in T.

Section 1.1 Properties of Othello

We previously described the Othello [25] data structure with a comprehensive evaluation of the algorithm. We summarize the properties of Othello as follows.

- An Othello data structure maintains the mapping $\tau: U \rightarrow C$, where $C = \{0, 1, \cdots, 2^l - 1\}$ and $l = \lceil \log_2 v + 1 \rceil$.
- Implementation of Othello entails (1) a pair of hash functions, $\langle h_a, h_b \rangle$, and (2) two arrays of l-bit integers, A and B. The lengths of the arrays, respectively denoted m_a and m_b, satisfy $2.67n \leq m_a + m_b < 4n$, where n is the number of *k*-mers. The functions and contents of the arrays are determined by the construction algorithm according to the keys and their corresponding categories. The time complexity of the construction algorithm is $O(n)$.
- An Othello built to map n *k*-mers to v categories requires at most $4n \lceil \log_2(v+1) \rceil$ bits of memory space.
- Given a *k*-mer s, its class information $\tau(s)$ is computed by $[h_A(s)] \oplus B[h_B(s)]$. Thus, querying a *k*-mer requires only two memory accesses and one XOR bit operation, making it extremely fast.

Section 1.2 On alien k-mer query of Othello

Let $\tau(s)$ be the category returned by querying a *k*-mer s on Othello $\boldsymbol{O}(S,V)$. If $s \in S$, then Othello guarantees that $\tau(s) = T(s)$. An alien *k*-mer $s' \notin S$ may be correctly

recognized as an alien if $\tau(s') \in C - V$; alternately, such a query may return a false positive if $\tau(s') \in V$. Our next goal is to analyze and bound the probability of Othello in recognizing alien k-mers.

Lemma 1: For any alien k-mer $s' \notin S$, a query on Othello $\boldsymbol{O}(S,V)$ returns an l-bit integer $\tau(s')$. For any integer x, the probability of $\tau(s') = x$ is denoted by p_x

$$p_x = \Pr[\tau(s') = x] = \sum_{t=0}^{2^l-1} a_t b_{x \oplus t}$$

Proof: Here a_x is the fraction of 0s in the array A of the Othello data structure, and b_x is the fraction of 0s in array B. The values of a_x and b_x are computed using the content stored in the memory of Othello. Lemma 1 is a direct application of a result presented in our previous work (MetaOthello [24], Section 2.2.3).

Lemma 2: Let $|S| = n$ for an Othello $\boldsymbol{O}(S,V)$ constructed with n elements. Let p_0 be the probability of an alien k-mer being assigned to category 0. p_0 satisfies $p_0 > 0.223$ as $n \to \infty$.

Proof: We prove Lemma 2 by giving an estimated lower bound on p_0. Array A of the Othello contains m_a elements. Each k-mer is mapped to an index of array A computed by $h_a(s)$, where h_a is a uniform random hash function. Assuming the number of k-mers, n, is large, the possibility of an index in A not being hit by any of the $h_a(s)$ values is

$$\lim_{n\to\infty} a_0 = \lim_{n\to\infty} \left(1 - \frac{1}{m_a}\right)^n = e^{-\frac{n}{m_a}}$$

An analogous statement holds for array B. Note that $m_a = 2^{\lceil \log_2 n \rceil}$ and $m_b = 2^{\lceil \log_2 \frac{4}{3} n \rceil}$. We have

$$1 < n\left(\frac{1}{m_a} + \frac{1}{m_b}\right) \le 1.5$$

$$p_{\text{Alien}} \ge p_0 = \sum_{x=0}^{2^l-1} a_x b_x > a_0 b_0 \to e^{-\frac{n}{m_a}} e^{-\frac{n}{m_b}} = e^{-n\left(\frac{1}{m_a}+\frac{1}{m_b}\right)} > e^{-1.5} = 0.223$$

Theorem 1: For any alien k-mer $s' \notin S$, the probability that s' is identified as "alien" by an Othello $\boldsymbol{O}(S,V)$ is given by:

$$P_{\text{Alien}} = p_0 + \sum_{x=\nu+1}^{2^l-1} p_x$$

We also have $p_{\text{Alien}} > 0.223$ as $|S| \to \infty$.

Proof: The probability that an alien k-mer falls into a class $x \in C - V$, denoted p_x, can be computed using the approach specified in Lemma 1. Note that $C - V = \{0, \nu + 1, \nu + 2, \cdots, 2^l - 1\}$, so that:

$$P_{\text{Alien}} = \Pr[\tau(s') \in C - V] = \Pr[\tau(s') = 0] + \sum_{x=\nu+1}^{2^l-1} \Pr[\tau(s') = x] = p_0 + \sum_{x=\nu+1}^{2^l-1} p_x$$

By Lemma 2, $P_{\text{Alien}} \ge p_0 > 0.223$ as $|S| \to \infty$.

Section 2. Encoding of *k*-mer occurrence map

We define the occurrence map of a k-mer as a binary vector recording the k-mer's presence or absence in each experiment. Given m experiments, the occurrence map can be stored using m bits, where 1 represents presence and 0 represents absence in a certain experiment. To minimize the storage requirement of these vectors, we have developed a hybrid encoding method that leverages one of three different encoding strategies depending on the occurrence frequency of a k-mer. Each k-mer is stored using the method that yields the shortest code. These encoding methods are detailed below:

- Value-list encoding. This method is used to compress occurrence maps associated with rare k-mers. For an m-bit occurrence map with exactly t 1s (representing presence in t out of m samples), we enumerate the t indices of these positions as a list. Each index is represented by t integers, each $\lceil \log_2 m \rceil$ bits long. This list can also be viewed as a $t\lceil \log_2 m \rceil$-bit integer. Value-list encoding is used when $t\lceil \log_2 m \rceil \le 64$.
- Delta-list encoding. This approach is employed for occurrence maps with a relatively larger number of 1s ($t\lceil \log_2 m \rceil > 64$). The m elements in the occurrence map can be considered as a succession of alternating subsequences of 0s and 1s. Thus, the map can be represented by a list of $2w + 1$ integers, $\langle x_1, y_1, x_2, y_2, \cdots, x_w, y_w, x_{w+1} \rangle$, representing the number of digits in each subsequence, where $x_1 \ge 0$, $x_{w+1} \ge 0$; $y_1, y_2, \cdots, y_w \ge 1$, $x_2, x_3, \cdots, x_w \ge 1$; and $x_1 + y_1 + x_2 + y_2 + \cdots + x_w + y_w + x_{w+1} = m$. The occurrence map can be reconstituted by enumerating x_1 0s, followed by y_1 1s, x_2 0s, y_2 1s, etc. For example, consider an occurrence map of $m = 20$ elements, 1110011...10, with 1s at indices 1, 2, 3, 6, 8, 9,…, 19. The corresponding delta-list representation is $\langle x_1 = 0, y_1 = 3, x_2 = 2, y_2 = 14, x_3 = 1 \rangle$.
  The $2w + 1$ integers from this first step are further encoded as positive integers. Multiple procedures exist for the second encoding step, the choice of which depends on the relative importance of minimizing encoding/decoding overhead versus maximizing the compression rate. To balance the time and memory complexity of encoding, as well as the storage overhead, we choose to encode the delta list as a hexadecimal stream. Each integer is converted to a hexadecimal value using the method

described in Table 1. We then concatenate the hexadecimal values into a single hexadecimal datum. For the delta list shown in the example, ⟨0, 3, 2,14,1⟩, the corresponding hexadecimal format is 0 × 8, 0 × B, 0 × A, 0 × 4E, 0 × 9. After concatenation, the final result is 0 × 8BA4E9.

- Bitmap encoding. Each occurrence bitmap is an *m*-bit value, with each bit coding the presence or absence information for one of the *m* samples. As this method requires more memory than other options, it is only used when a value-list or delta-list cannot generate a more efficient encoding.

Section 3. Construction of SeqOthello

Section 3.1 Construction algorithm

Construction of a SeqOthello data structure requires as input a list of *k*-mer files, each containing the set of *k*-mers extracted from reads associated with a distinct RNA-seq experiment. Currently the *k*-mer file is generated by applying Jellyfish to fastq files.

Step 1: Assembling the occurrence map of each *k*-mer in the collection of experiments to be indexed The goal of step 1 is to determine each *k*-mer's presence/absence information across all experiments. This task requires the integration of *k*-mers from all *k*-mer files, but simultaneous file access is time-consuming and not allowed by many operating systems. Instead, we employ a strategy similar to merge sort. We first obtain *k*-mer occurrence maps for small groups of experiments, where each group contains approximately 50 samples. These intermediate occurrence maps are encoded as delta lists, which significantly reduces file sizes. The groups are then merged to obtain the *k*-mer occurrences across all experiments. After SeqOthello is constructed, the group files generated at this step are no longer needed. However, as these files are orders of magnitude smaller than the original *k*-mer files, they can be stored to support update of the SeqOthello structure.

Step 2: Assignment of *k*-mer occurrence maps to buckets We next divide the entire set of *k*-mers into disjoint buckets based on their occurrence maps using the following principles: (1) Occurrence maps within the same bucket should be generated by the same encoding approach; (2) the lengths of encoded occurrence maps within the same bucket should have limited variation; and (3) the total size of the encoded occurrence maps within each bucket should not exceed a specified threshold (by default, 128 MB).

Given a maximum bucket size, we define the range of encoding lengths for each bucket prior to allocating *k*-mers. Note that the distribution of *k*-mer encoding lengths is unknown prior to construction. To avoid multiple iterations over all *k*-mers during bucket assignment, we designed a sampling-based approach to estimate the range of encoding lengths. The goal is to set an open upper bound n_{t+1} and closed lower bound n_t so that *k*-mers with encoding lengths in the range $[n_t, n_{t+1})$ are assigned to each bucket *t*. We select 10 million *k*-mers, which is approximately 0.1% of the *k*-mers present over all experiments, and let L_i be the estimated number of *k*-mers with encoding length equal to *i*. Starting from $t = 1$ and $n_1 = 1$, we greedily select the maximum index n_{t+1} so that $n_t L_{n_t} + (1+n_t)L_{1+n_t} + \cdots + (n_{t+1}-1)L_{n_{t+1}-1} \le 128\text{M}$. Once the number of buckets and their ranges of encoding lengths are determined, the construction algorithm will iterate over each *k*-mer, assigning it to the appropriate bucket in accordance with the encoding length of its occurrence map. The encoded occurrence maps are further compressed by gzip when the final structure is stored as a file.

Step 3: Establish *k*-mer mapping using Othello During step 2, SeqOthello maintains the list of *k*-mers and their corresponding encoded occurrence maps in each bucket. Once the *k*-mer assignment is completed in the bucket, an Othello will be established to record the mapping between *k*-mers and the locations of their occurrence maps. Once the buckets are finalized, a root Othello is constructed to record the mapping between the entire set of *k*-mers and their bucket IDs.

SeqOthello also maintains an .xml file to store metadata associated with the data structure, which includes basic information about the experiments and information necessary for the query algorithm to interpret the data file.

Table 1 Hexadecimal encoding for integer values in the delta-list encoding

Integer value *z*	Encoded binary representation	Hexadecimal value	Encoded length in bits
$0 \le z < 8$	$(1xxx)_2$	0x8 | *z*	4
$8 \le z < 64$	$(01xxxxxx)_2$	0x40 | *z*	8
$64 \le z < 512$	$(001xxxxxxxxx)_2$	0x200 | *z*	12
$512 \le z < 4096$	$(0001xxxxxxxxxxxx)_2$	0x1000 | *z*	16
$4096 \le z$	$(0000xxxxxxxxxx\ldots)_2$	0x0000 | *z*	32

Section 3.2 Optimization for k-mers that appear in only one experiment

The prevalence of individual k-mers varies dramatically, with plots often exhibiting a U- or L-shaped distribution (Additional file 1: Figure S1). Note that the number of k-mers present in only one experiment is relatively large compared to k-mers with higher frequencies. We apply the following approach to improve the efficiency and accuracy of SeqOthello.

Instead of storing all k-mers with single occurrence in a level-2 bucket, we encode them directly in the root Othello. Let E be the set of experiments indexed by SeqOthello, identified by integers $\{1, 2, \cdots, |E|\}$. Let B be the set of buckets identified by integers $\{|E| + 1, |E| + 2, \cdots, |E| + |B|\}$. The root Othello records the mapping between k-mer set S and $E \cup B$. For any k-mer s, if the query result on the first level $\tau(s) \in \{1, 2, \cdots, |E|\}$, SeqOthello will report that s is present in the experiment with index $\tau(s)$; if $\tau(s) = |E| + b$ for some integer $b \in \{1, 2, \cdots, |B|\}$, then $\tau(s) \in B$ and the query process will continue into the bucket with index b on the bottom layer of SeqOthello.

Section 3.3 Insertion of new experiments into SeqOthello

If the group files generated at step 1 have been retained, the insertion of new experiments to SeqOthello is quite fast, especially for batch update. The process involves merging newly inserted experiments with the existing group files, and then repeating steps 2 and 3 of the above construction algorithm. The entire update requires only a few hours to complete.

Section 4. The probability of false-positive *k*-mer query with SeqOthello

SeqOthello maintains a mapping from a large set of k-mers to their occurrence maps. However, due to the nature of Othello being a minimal perfect hashing classifier, querying of an alien k-mer (i.e., k-mer that does not exist in any of the samples) with SeqOthello may afford a false report of its presence in one or more RNA-seq experiments. Here, we analyze the likelihood of such a false report.

Section 4.1 Notations

In reference to SeqOthello, we use the notation ${}^{\text{Root}}\boldsymbol{O}(S, V)$ to denote the root-level Othello. ${}^{\text{Root}}\boldsymbol{O}(S, V)$ records the mapping between a k-mer in S and its assignment either to a single experiment or to a second-level bucket in $V = E \cup B$.

For any bucket $b \in B$, we use the notation ${}^{b}\boldsymbol{O}(S_b, V_b)$ to denote the associated Othello, where ${}^{b}\boldsymbol{O}(S_b, V_b)$ stores the mapping between a k-mer in S_b and its occurrence map index in V_b. Thus, S_b is the set of k-mers that are assigned to bucket b and $V_b = \{1, 2, \cdots, v_b\}$ is the list of indices for encoded occurrence maps in bucket b.

We list the primary notation used in the following analysis in Table 2.

Section 4.2 Probability of alien k-mer recognition and false-positive presence

Let s' be an alien k-mer and $\tau(s')$ be the result returned when querying s' on the root Othello. Then, $\tau(s')$ falls into one of the following three categories:

A. $\tau(s') \notin V$, where $V = E \cup B$. This k-mer will be identified as alien, and SeqOthello will report its absence from the database. The probability of this result is ${}^{\text{root}}P_{\text{Alien}}$, which can be calculated according to Theorem 1.
B. $\tau(s') \in E$. Such a k-mer will be reported falsely as existing in the experiment identified by $\tau(s')$. For any experiment $e \in E$, the probability of returning e as the result of querying an alien k-mer has a probability ${}^{\text{root}}p_e$, which can be calculated based on Lemma 1.
C. $\tau(s') \in B$. In this case, the query process would continue into the bucket b identified by $\tau(s')$. This circumstance occurs with probability ${}^{\text{root}}p_{|E|+b}$. Inside the bucket b, the query ${}^{b}\tau(s')$ will result in one of two scenarios:
   1) ${}^{b}\tau(s') \notin V_b$. In this case, s' is identified as alien in bucket b with probability ${}^{b}P_{\text{Alien}}$, which is P_{Alien} for the Othello ${}^{b}\boldsymbol{O}(S_b, V_b)$.
   2) ${}^{b}\tau(s') \in V_b$. Here s' is mapped falsely to a location storing the occurrence map of a different k-mer. A calculation follows for the probability of this outcome.

   Assume there are v_b encoded occurrence maps stored in bucket b, namely $W_1, W_2, \cdots, W_{v_b}$. We use the notation $W_{t,e} \in \{0, 1\}$ to denote the presence/

Table 2 A summary of notations used in Section 4

${}^{\text{Root}}\boldsymbol{O}(S, V)$	Othello at the root of SeqOthello
${}^{b}\boldsymbol{O}(S_b, V_b)$	Othello of the bucket b
E	Set of RNA-seq experiments
B	Set of buckets
W_t	tth occurrence map in a bucket b
${}^{\text{SeqOthello}}P_{\text{Alien}}$	Probability of an alien k-mer being recognized as alien by SeqOthello
${}^{\text{SeqOthello}}P(e)$	Probability of an alien query returning experiment e
${}^{\text{root}}p_x$	Probability that query of an alien k-mer on the root Othello $\tau(s')$ returns x
${}^{b}p_x$	Probability that query of an alien k-mer on the Othello in bucket b returns ${}^{b}\tau(s)$ value x

absence information for experiment e stored in the tth occurrence map. Here, $W_{t,\,e} = 1$ indicates that the k-mer associated with occurrence map W_t is marked as "present" in experiment e; $W_{t,\,e} = 0$ indicates it is marked as "not present" in experiment e.
Note that a query on bucket b returns the occurrence map with index ${}^{b}\tau(s')$, namely $W_{{}^{b}\tau(s')}$. For any experiment e, $1 \le e \le \mid E \mid$, if $W_{{}^{b}\tau(s'),\boldsymbol{e}} = 1$, then the query result would indicate falsely that s' is present in experiment e. We use the notation ${}^{b}P(e)$ to denote the probability of the query on bucket b yielding $W_{{}^{b}\tau(s'),\boldsymbol{e}} = 1$. ${}^{b}P(e)$ is equal the probability of ${}^{b}\tau(s')$ returning any index x such that the xth occurrence map W_x satisfies $W_{x,\,e} = 1$:

$$
\begin{aligned}
{}^{b}P(e) &= \Pr\left[W_{{}^{b}\tau(s'),e} = 1\right] \\
&= \sum_{x \in V_b} \Pr\left[{}^{b}\tau_{(s')} = x \wedge W_{x,e} = 1\right]
\end{aligned}
$$

Noting that $W_{x,\,e} \in \{0, 1\}$,

$$
{}^{b}P(e) = \sum_{x \in V_b} {}^{b}p_x W_{x,e}
$$

Computing ${}^{b}p_x$ for all $x \in V_b$ using Lemma 1 requires $O((2^{l_b})^2 |V_b|)$ computation, which becomes infeasible when l_b is large. Hence, we use an alternative approach to estimate the ${}^{b}p_x$ values when $l \ge 12$. Lemma 2 indicates that the value of ${}^{b}p_0$ is significantly larger than ${}^{b}p_x$values for $x \ne 0$. We also observe that the values for ${}^{b}p_x$ are similar for any $x \ne 0$ and $x < 2^{l_b}$ in the same bucket b. We therefore use the average value of ${}^{b}p_x$ over $x \ne 0$, denoted by $\overline{{}^{b}p_{x\neq 0}}$, to replace individual ${}^{b}p_x$ values:

$$
\overline{{}^{b}p_{x\neq 0}} = \frac{1}{2^{l_b} - 1}\left(1 - {}^{b}p_0\right)
$$

Hence,

$$
\begin{aligned}
{}^{b}P(e) &= \sum_{x \in V_b} {}^{b}p_x W_{x,e} \rightarrow \sum_{x \in V_b} \overline{{}^{b}p_{x\neq 0}} W_{x,e} \\
&= \overline{{}^{b}p_{x\neq 0}} \sum_{x \in V_b} W_{x,e} = \frac{\left(1 - {}^{b}p_0\right)}{2^{l_b} - 1} \sum_{x \in V_b} W_{x,e}
\end{aligned}
$$

Here, $\sum_{x \in V_b} W_{x,e}$ is the number of encoded occurrence maps in bucket b in which the associated k-mer is marked to be present in experiment e.

For an alien k-mer s', the query on SeqOthello may return a false presence in experiment e if $\tau(s')$ falls in category B, a circumstance which occurs with probability ${}^{root}p_e$. Otherwise, if $\tau(s')$ satisfies circumstance C.2, the query yields an occurrence map in which experiment e is marked as positive with probability ${}^{b}P(e)$. Hence, the probability of an alien k-mer query on the two-level SeqOthello yielding a false-positive presence in experiment e is:

$$
{}^{SeqOthello}P(e) = {}^{root}p_e + \sum_{b=1}^{|B|} {}^{root}p_{|E|+b} \cdot {}^{b}P(e)
$$

On the other hand, an alien k-mer has a very good likelihood of being recognized as alien if $\tau(s')$ satisfies circumstance A, or falls in circumstance C and is subsequently identified under C.1. Taken together, the overall probability of SeqOthello identifying the k-mer as alien is:

$$
{}^{SeqOthello}P_{\text{Alien}} = {}^{root}P_{\text{Alien}} + \sum_{b=1}^{|B|} {}^{root}p_{|E|+b} \cdot {}^{b}P_{\text{Alien}}
$$

We present a numerical estimation of various probabilities based on the distribution of k-mer occurrences as well as the SeqOthello structures constructed for the two datasets used in this paper. The results are given in Table 3 below.

Section 4.4 Error rate of a SeqOthello sequence query

SeqOthello executes sequence query by making individual k-mer queries extracted from the sequence. The probability of returning false-positive k-mer hits is low and can be computed as ${}^{\text{SeqOthello}}P(e)$. Let $X(e)$ be the number of false positives for experiment e returned over w alien k-mer queries. Then, $X(e)$ follows the binomial distribution $Binomial(w, {}^{\text{SeqOthello}}P(e))$. Note that the query result for transcript query is reported as the fraction of present k-mers for each sample, and $X(e)$ false-positive k-mers will result in an error rate of $\frac{X(e)}{w}$. Note that the $\frac{X(e)}{w}$ is usually 0. The probability of $\frac{X(e)}{w}$ being large enough to affect the query result is very low, only occurring when multiple k-mer queries return the same false-positive experiments. For example, for $w = 50$ and $P(e) = 0.0084$, the probability of $X(e) > 2$ is 1.15×10^{-5}. Thus, SeqOthello returns the query result with error rate $\delta = \frac{X(e)}{w} > \frac{2}{50} = 4\%$ with probability 1.15×10^{-5}, which is much lower than the probability of a single error.

Table 3 Estimated probability values computed on SeqOthello constructed for human and TCGA datasets

	SRA	TCGA
$\mid \boldsymbol{E} \mid$: number of experiments	2652	10,113
$\mid \boldsymbol{B} \mid$: number of buckets	105	127
${}^{\text{SeqOthello}}P_{\text{Alien}}$	0.532440	0.551722
${}^{\text{SeqOthello}}P(e)$, average over all experiments	0.000840	0.000606
standard deviation of ${}^{\text{SeqOthello}}P(e)$, across all experiments	0.000684	0.000173

Table 4 SBT, SSBT, and SBT-AS version information

Algorithm	URL	Version
SBT	https://github.com/Kingsford-Group/bloomtree	f7986e4511189cb781b4e3517626b396fb11eefa
SSBT	https://github.com/Kingsford-Group/splitsbt	0fe43f4c0de7a0a452486a252a5e317862c2af45
SBT-AS	https://github.com/medvedevgroup/bloomtree-allsome	383d23f17d5537a0abf1436e5d04795ef91950b3

Section 5. Performance comparison

Section 5.1 System configuration

All comparison tests with SBT, SSBT, and SBT-AS were conducted on a Linux OS (RHEL) server with Quad Intel E5-4640 8 core (Sandy Bridge) @ 2.4 GHz processors, 512 GB of 1600 Mhz RAM, and 4 × 1 TB local (internal) NLSAS disk.

Section 5.2 Versions and parameters for SBT, SSBT, and SBT-AS

SBT, SSBT, and SBT-AS versions used in the evaluation are provided in Table 4.

Section 6. Estimation of typical fusion-detection processing time

An alternative approach to test whether a fusion event occurs in a sample is by checking whether there are reads that can be properly aligned to the fusion sequence. To estimate the performance of querying fusion events detection using aligner-based approach, we built a STAR [32] index for the 11,658 tier-1 fusion transcripts curated by TCGA Fusion Gene Database and aligned the reads against it. The average speed for STAR to process 1 million paired-end reads using a 16-core CPU is benchmarked at 0.11 min using ten random samples of TCGA datasets. The TCGA RNA-seq dataset is estimated to contain a total of 660 billion reads. Thus, it will cost about 54 days of computation using 16-core CPUs to search through all the TCGA RNA-seq dataset for known fusion detection regardless of the alignment accuracy.

A study [33] published recently by Kumar et al. details a comprehensive comparison of 12 fusion-detection algorithms, including FusionHunter [34], FusionMap [35], Bellerophontes [36], MapSplice [37], Chimerascan [38], TopHat-Fusion [39], BreakFusion [40], SOAPfuse [41], JAFFA [42], nFuse [43], EricScript [44], and FusionCatcher [45]. The authors reported that it requires 120 to 3845 min for current tools to process a dataset of 70 million paired-end reads, averaging between 1.71 and 54 min per million reads. The TCGA RNA-seq dataset is estimated to contain 660 billion reads. Taking the fastest processing speed regardless of accuracy, we estimate that it costs 785 days of computation to process all the TCGA data for fusion detection using standard tools.

Additional files

Additional file 1: Figure S1. The histograms of *k*-mer occurrence frequencies in two human RNA-Seq datasets. (PDF 154 kb)

Additional file 2: Details of SRA samples used for performance comparison between SeqOthello and other SBT-based methods. (XLSX 53 kb)

Additional file 3: Table S1 Performance comparison on index construction. (PDF 13 kb)

Additional file 4: Table S2. Performance comparison on small batch query. (PDF 55 kb)

Additional file 5: Figure S2. A Venn diagram showing the accuracy in querying human transcriptomic *k*-mers totaling 120,044,842 from experiment SRR925711. (PDF 143 kb)

Additional file 6: Details of TCGA samples used to contruct SeqOthello as well as fusion occurrences detected by querying the index. (XLSX 3612 kb)

Additional file 7: Figure S3. An illustration of fusion junction sequence constructed for fusion query using SeqOthello. (PDF 107 kb)

Additional file 8: Review history. (DOCX 166 kb)

Review history

The review history for this manuscript is included as Additional file 8.

Funding

This work was supported by US National Science Foundation [award grant number 1054631 to J.L., CNS-1717948 and CNS-1750704 to C.Q.] and National Institutes of Health [grant number P30CA177558 and 1UL1TR001998-01 to J.L.] The Seven Bridges Cancer Genomics Cloud [47] has been funded in whole or in part with Federal funds from the National Cancer Institute, National Institutes of Health, Contract No. HHSN261201400008C and ID/IQ Agreement No. 17X146 under Contract No. HHSN261201500003I.

Availability of data and materials

SeqOthello is an Open Source software under GPL 3.0 License. The source code of SeqOthello is available at Github repository, <https://github.com/LiuBioinfo/SeqOthello>. The SeqOthello version, and the scripts used to build and query the SeqOthello mapping are also available on Zenodo [46].
The results shown here are partially based upon data publicly available at sequence read archive (SRA) [1] and data generated by the TCGA [2] Research Network. The detailed list of datasets used for evaluation are provided in Additional file 2 and Additional file 6 respectively.
All final data generated or analyzed during this study are included in this published article and its supplementary information files.

Authors' contributions

JL conceived and directed the project. JL, YY, and XL designed the computational algorithm. YY implemented the algorithm and conducted the theoretical analysis. YY, JPL, and YZ conducted the performance evaluation and comparison. XL interpreted the fusion calling results. JL and YY drafted the paper with input from all authors. All authors read and approved the final manuscript.

Ethics approval and consent to participate

No ethical approval was required for this study. All utilized public data sets were generated by other organizations that obtained ethical approval.

Consent for publication

Not applicable.

Competing interests
The authors declare that they have no competing interests. Erik Lehnert was an employee of Seven Bridges, Inc.

Publisher's Note

Author details
[1]Department of Computer Science, University of Kentucky, 301 Rose St, Lexington, KY 40508, USA. [2]Seven Bridges Genomics Inc, 1 Main St, 5th Floor, Suite 500, Cambridge, MA 02142, USA. [3]Department of Computer Engineering, University of California Santa Cruz, 1156 High Street, Santa Cruz, CA 95064, USA.

Received: 18 February 2018 Accepted: 11 September 2018
Published online: 19 October 2018

References
1. National Centre for Biotechnology Information. SRA: sequence read archive. NCBI Handout Ser 4 (2015).
2. TCGA. The Cancer Genome Atlas; 2015. p. 2015.
3. International Cancer Genome Consortium. International Cancer Genome Consortium. Available at: http://icgc.org/. (Accessed 5 Nov 2017).
4. Barrett T, et al. NCBI GEO: archive for functional genomics data sets—update. Nucleic Acids Res. 2012;41.D1:D991–D995.
5. Pertea M, Kim D, Pertea GM, Leek JT, Salzberg SL. Transcript-level expression analysis of RNA-seq experiments with HISAT, StringTie and Ballgown. Nat Protoc. 2016;11:1650–67.
6. NCBI Resource Coordinators. Database resources of the National Center for biotechnology information. Nucleic Acids Res. 2017;45:D12–7.
7. Petryszak R, et al. Expression Atlas update--a database of gene and transcript expression from microarray- and sequencing-based functional genomics experiments. Nucleic Acids Res. 2014;42:D926–32.
8. Collado-Torres L, et al. Reproducible RNA-seq analysis using recount2. Nat Biotechnol. 2017;35:319–21.
9. Nellore A, et al. Human splicing diversity and the extent of unannotated splice junctions across human RNA-seq samples on the sequence read archive. Genome Biol. 2016;17:266.
10. Solomon B, Kingsford C. Fast search of thousands of short-read sequencing experiments. Nat Biotechnol. 2016;34:300–2.
11. Solomon B, Kingsford C. Improved search of large transcriptomic sequencing databases using Split sequence bloom trees Brad. Res Comput Mol Biol. 2017;10229:257–71.
12. Sun C, Harris RS, Chikhi R, Medvedev P. Allsome sequence bloom trees. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) 10229 LNCS; 2017. p. 272–86.
13. Bloom BH. Space/time trade-offs in hash coding with allowable errors. Commun ACM. 1970;13:422–6.
14. Broder A, Mitzenmacher M. Network applications of bloom filters: a survey. Internet Math. 2004;1:485–509.
15. Pandey P, et al. Mantis: a fast, small, and exact large-scale sequence search index. *bioRxiv*. 2017:217372. https://doi.org/10.1101/217372.
16. Holley G, Wittler R, Stoye J. Bloom Filter Trie: an alignment-free and reference-free data structure for pan-genome storage. Algorithms Mol Biol. 2016;11:3.
17. Dolle DD, et al. Using reference-free compressed data structures to analyze sequencing reads from thousands of human genomes. Genome Res. 2017; 27:300–9.
18. 1000 Genomes Project Consortium, et al. A global reference for human genetic variation. Nature. 2015;526:68–74.
19. Grabherr MG, et al. Full-length transcriptome assembly from RNA-Seq data without a reference genome. Nat Biotechnol. 2011;29:644–52.
20. Haas BJ, et al. De novo transcript sequence reconstruction from RNA-seq using the trinity platform for reference generation and analysis. Nat Protoc. 2013;8:1494–512.
21. Li B, et al. Evaluation of de novo transcriptome assemblies from RNA-Seq data. Genome Biol. 2014;15.12:553.
22. Shannon CE. A mathematical theory of communication. Bell Syst Tech J. 1948;27:379–423.
23. Borda M. Fundamentals in information theory and coding. J Chem Inf Model. 2011;53.
24. Liu X, et al. A novel data structure to support ultra-fast taxonomic classification of metagenomic sequences with *k*-mer signatures. Bioinformatics. 2017. https://doi.org/10.1093/bioinformatics/btx432.
25. Yu Y, Belazzougui D, Qian C, Zhang Q. Memory-efficient and ultra-fast network lookup and forwarding using Othello hashing. IEEE/ACM Trans Networking. 2018:1–14. https://doi.org/10.1109/TNET.2018.2820067.
26. Marçais G, Kingsford C. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. Bioinformatics. 2011;27:764–70.
27. Harrow J, et al. GENCODE: the reference human genome annotation for the ENCODE project. Genome Res. 2012;22:1760–74.
28. Yoshihara K, et al. The landscape and therapeutic relevance of cancer-associated transcript fusions. Oncogene. 2015;34:4845–54.
29. Torres-García W, et al. PRADA: pipeline for RNA sequencing data analysis. Bioinformatics. 2014;30:2224–6.
30. Majewski BS, Wormald NC, Havas G, Czech ZJ. A family of perfect hashing methods. Comput J. 1996;39:547–54.
31. Yu Y, Li X, Qian C. SDLB: A scalable and dynamic software load balancer for fog and mobile edge computing. In Proceedings of the Workshop on Mobile Edge Communications (MECOMM) 55–60. Los Angeles: ACM Press; 2017.
32. Dobin A, et al. STAR: ultrafast universal RNA-seq aligner. Bioinformatics. 2013;29:15–21.
33. Kumar S, Vo AD, Qin F, Li H. Comparative assessment of methods for the fusion transcripts detection from RNA-Seq data. Sci Rep. 2016;6:21597.
34. Li Y, Chien J, Smith DI, Ma J. FusionHunter: identifying fusion transcripts in cancer using paired-end RNA-seq. Bioinformatics. 2011;27:1708–10.
35. Ge H, et al. FusionMap: detecting fusion genes from next-generation sequencing data at base-pair resolution. Bioinformatics. 2011;27:1922–8.
36. Abate F, et al. Bellerophontes: an RNA-Seq data analysis framework for chimeric transcripts discovery based on accurate fusion model. Bioinformatics. 2012;28:2114–21.
37. Wang K, et al. MapSplice: accurate mapping of RNA-seq reads for splice junction discovery. Nucleic Acids Res. 2010;38.18:e178–e178.
38. Iyer MK, Chinnaiyan AM, Maher CA. ChimeraScan: a tool for identifying chimeric transcription in sequencing data. Bioinformatics. 2011;27:2903–4.
39. Kim D, Salzberg SL. TopHat-Fusion: an algorithm for discovery of novel fusion transcripts. Genome Biol. 2011;12.8:R72.
40. Chen K, et al. Breakfusion: targeted assembly-based identification of gene fusions in whole transcriptome paired-end sequencing data. Bioinformatics. 2012;28:1923–4.
41. Jia W, et al. SOAPfuse: an algorithm for identifying fusion transcripts from paired-end RNA-Seq data. Genome Biol. 2013;14.2:R12.
42. Davidson NM, Majewski IJ, Oshlack A. JAFFA: high sensitivity transcriptome-focused fusion gene detection. Genome Med. 2015;7.1:43.
43. McPherson A, et al. NFuse: discovery of complex genomic rearrangements in cancer using high-throughput sequencing. Genome Res. 2012;22:2250–61.
44. Benelli M, et al. Discovering chimeric transcripts in paired-end RNA-seq data by using EricScript. Bioinformatics. 2012;28:3232–9.
45. Nicorici D, et al. FusionCatcher - a tool for finding somatic fusion genes in paired-end RNA-sequencing data. bioRxiv. 2014. https://doi.org/10.1101/011650.
46. Yu, Y. et al. (2018). SeqOthello: query over RNA-seq experiments at scale (version 1.0.0). Zenodo. https://doi.org/10.5281/zenodo.1240556.
47. Lau, et al. The Cancer Genomics Cloud: collaborative, reproducible, and democratized—a new paradigm in large-scale computational research. Cancer Res. 2017;77(21):e3–6. https://doi.org/10.1158/0008-5472.CAN-17-0387.