BIOINFORMATICS ORIGINAL PAPER Vol. 27 no. 24 2011, pages 3348–3355
doi:10.1093/bioinformatics/btr560

Sequence analysis

Advance Access publication October 8, 2011

KABOOM! A new suffix array based algorithm for clustering expression data

Scott Hazelhurst[1,*] and Zsuzsanna Lipták[2,*]

[1]Wits Bioinformatics, School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg, Private Bag 3, 2050 Wits, South Africa and [2]Università degli Studi di Salerno, Dipartimento di Informatica, Via Ponte don Melillo, 1, 84084 Fisciano, Italy

Associate Editor: John Quackenbush

ABSTRACT

Motivation: Second-generation sequencing technology has reinvigorated research using expression data, and clustering such data remains a significant challenge, with much larger datasets and with different error profiles. Algorithms that rely on all-versus-all comparison of sequences are not practical for large datasets.

Results: We introduce a new filter for string similarity which has the potential to eliminate the need for all-versus-all comparison in clustering of expression data and other similar tasks. Our filter is based on multiple long exact matches between the two strings, with the additional constraint that these matches must be sufficiently far apart. We give details of its efficient implementation using modified suffix arrays. We demonstrate its efficiency by presenting our new expression clustering tool, wcd-express, which uses this heuristic. We compare it to other current tools and show that it is very competitive both with respect to quality and run time.

Availability: Source code and binaries available under GPL at http://code.google.com/p/wcdest. Runs on Linux and MacOS X.

Contact: scott.hazelhurst@wits.ac.za; zsuzsa@cebitec.uni-bielefeld.de

Supplementary Information: Supplementary data are available at *Bioinformatics* online.

Received on June 15, 2011; revised on August 26, 2011; accepted on September 20, 2011

1 INTRODUCTION

The clustering of expressed sequence tags (ESTs) and other gene expression data continues to be a major challenge in bioinformatics. The emergence of new sequencing technologies such as pyrosequencing, collectively referred to as *second-generation sequencing* (Pop and Salzberg, 2008; Robison, 2010), has recently reinvigorated studies using expression data. Second-generation sequencing provides the opportunity to study the transcriptomes of organisms for which good quality genomes are not known. However, new computational challenges have emerged, with much larger datasets, shorter sequence length and new error profiles (Schwartz and Waterman, 2009).

In expression clustering, we start with a large set of cDNA sequences, typically 10^5 or more, which have been derived from transcriptomic data in a laboratory process (commonly, these sequences are referred to as ESTs). The goal is to find a partitional clustering such that sequences derived from the same gene are members of the same cluster. Expression clustering can broadly be divided into two classes: (i) clustering for which a reference genome is known (supervised clustering) and (ii) clustering for which a reference genome is not known (also called *ab initio* or *de novo* clustering). In this article, we focus on the latter class.

Typically, single linkage clustering is used for expression data: if two sequences are similar, their clusters are merged. Within this approach, different similarity measures can be used.

Traditionally, edit distance/alignment has been used to define similarity between sequences. However, alignment-free measures are increasingly being adopted, such as q-gram distance (Ukkonen, 1992) or d^2 (Torney *et al.*, 1990). These define similarity between sequences with respect to the multiplicity of substrings (subwords) of a fixed, usually small, length. Because of effects such as alternative splicing, in expression clustering typically a local similarity of a predefined length is sought. For two sequences of length m to be regarded as similar, it suffices to find a pair of similar windows. Using subword-based measures, it is possible to compute the maximum similarity between all pairs of windows of a fixed length in time $O(m^2)$ [similarly, computation of an optimal local alignment score takes $O(m^2)$ time].

EST clustering algorithms that use subword-based distance measures rather than alignment methods have proved successful (Hazelhurst *et al.*, 2008; Kalyanaraman *et al.*, 2002; Miller *et al.*, 1999). However, with the new and much larger datasets, computation time is still an issue: given n EST sequences, with average length m, computing all pairwise similarities requires $\Theta(n^2m^2)$ time. For real datasets this is prohibitive, at least without massive parallelism.

Much work has gone into breaking these complexity limits. Filtering heuristics have been very successful. They test two strings in linear time to see whether they are likely to be similar, before a more expensive comparison is done. In practice, these heuristics have sped up clustering by orders of magnitude. However, the algorithms still remain quadratic in the number of sequences.

In this article, we introduce the KABOOM filter, which greatly reduces the number of candidate pairs without compromising on clustering quality. This heuristic passes a pair of sequences if they share a given number of common words (substrings) of a given length, occurring at least a given distance apart. We also give details of its efficient implementation, which uses a modified suffix array.

Contribution: our contribution is 2-fold:

(1) We introduce a new heuristic filter for sequence similarity.

*To whom correspondence should be addressed.

(2) We demonstrate a method which, at least for real-world data, implements this heuristic and eliminates the need for all-versus-all comparison.

The computation of our heuristic runs in time which depends on the multiplicities of words of the given length within the set to be clustered. Although in the worst case, the computation of our heuristic could take quadratic time in n, the number of sequences, the cost on real-life data is significantly subquadratic. On the other hand, the filtration rate of the KABOOM heuristic is impressive: on all our experimental data, the number of pairwise comparisons was reduced to far below 1%.

We present extensive experimentation results which show how the KABOOM heuristic has sped up our previous EST clustering tool, WCD (Hazelhurst, 2008; Hazelhurst *et al.*, 2008), without compromising clustering quality. WCD uses the subword based dissimilarity measure d^2 to cluster sequences, shown to be competitive with or even superior to many existing tools, both with respect to running time and clustering quality (Hazelhurst *et al.*, 2008), where we reported comparison studies with ESTate (Slater, 2000), xsact (Malde *et al.*, 2003), PaCE (Kalyanaraman *et al.*, 2002) and the assembler tool CAP3 (Huang and Madan, 1999). Here, we compare our new tool, called WCD-EXPRESS, to WCD and to two other EST clustering tools, TGICL (Pertea *et al.*, 2003) and the recent tool PEACE (Rao *et al.*, 2010), which claims to have better performance than WCD. WCD-EXPRESS outperforms them both with respect to run time and produces at least as good quality. We show applicability to Sanger style data as well as to 454 and Illumina data (second-generation sequencing). We believe that the new filter has the potential to achieve similar speedups for other tools which rely on pairwise comparison of a set of sequences.

Related work: EST clustering tools based on common words include PaCE (Kalyanaraman *et al.*, 2002), `xsact` (Malde *et al.*, 2003), QUASAR (Burkhardt *et al.*, 1999), more recently our tool WCD (Hazelhurst *et al.*, 2008) and PEACE (Rao *et al.*, 2010). Several of these use suffix trees or suffix arrays for finding common words.

PaCE and WCD both explore a heuristic of finding candidate matches based on a shared common word to avoid all-versus-all comparison. The basic idea is to choose a length k and to do a full comparison only on those pairs of sequences that share at least one word of length k. These pairs can be found efficiently using a suffix array or suffix tree, which allows to identify all sequences that contain a given k-word.

The KABOOM heuristic adapts this idea to sequence pairs which share *several* exact matches. Our approach is closest to that of `xsact` (Malde *et al.*, 2003), which also uses multiple common words as a criterion for similarity. Unlike in KABOOM, `xsact` uses variable size common words, which must occur in the same order in both strings. However, the biggest difference is in the implementation. The authors only report results on one fairly small dataset; in our earlier work, we found the tool was very slow on large sets, and it did not scale (Hazelhurst *et al.*, 2008). This article shows how a related approach can be efficiently implemented and integrated in a well-established EST clustering system.

2 PRELIMINARIES

We start with some formal definitions. A *string* (or *word* or *sequence*) over a finite alphabet Σ is a finite sequence $s=s_1\ldots s_n$ of characters from $\Sigma=\{A,C,G,T\}$. Σ^* is the set of all strings over Σ. For a string $s=s_1\ldots s_n\in\Sigma^*$, we denote by $|s|$ its length n. For two strings $s,w\in\Sigma^*$, where $s=s_1\ldots s_n$ and $w=w_1\ldots w_k$, w is a *substring*, or *subword* of s, denoted $w\sqsubseteq s$, if there exists an index $1\leq i\leq n$ such that $s_i\ldots s_{i+k-1}=w_1\ldots w_k$. Such an index i is called an *occurrence* of w in s. A string of length k is also referred to as a k-*word*.

2.1 Clustering based on pairwise string similarity

Given a set S of strings over $\{A,C,G,T\}$ (sequences derived from expression data), we want to find a clustering, i.e. a partition, of S such that two sequences end up in the same cluster if and only if they have been derived from the same gene. Since it is difficult to detect when two sequences have been derived from the same gene, we formally state our problem as:

Problem statement (*Expression Clustering*). Given a set S of sequences over the alphabet $\Sigma=\{A,C,G,T\}$, compute a clustering $\mathcal{C}=\{C_1,\ldots,C_r\}$ of S such that (i) $\bigcup_{i=1}^{r}C_i=S$, and for $i\neq j$, $C_i\cap C_j=\emptyset$; and (ii) for all $s,t\in S$, if s and t are similar according to the given similarity or dissimilarity measure, then s and t are in the same cluster.

This is the definition for *ab initio, partitional, single-linkage* clustering. For definitions of other types of clustering, see e.g. (Jain *et al.*, 1999). Condition (i) states that $\mathcal{C}$ is a partition of S. Dissimilarity measures commonly used for string comparison in EST clustering include the *edit distance* (Levenshtein, 1966), *q-gram distance* (Ukkonen, 1992) and d^2 (Torney *et al.*, 1990). Usually, one decides on a threshold $\theta\in\mathbb{R}^+$, and then two sequences s,t are said to be similar if $d(s,t)\leq\theta$, where d is the dissimilarity measure.

In our tool, we use the dissimilarity measure d^2. We denote by $\text{mult}(w,s)$ the number of occurrences of the word w in s.

Definition 2.1 (*Windowed* d^2). Let $s,t\in\Sigma^*$, Let q (the word size) and λ (the window length) be two positive integers. Then $d^2(s,t)=\min\{\sum_{w\in\Sigma^q}\left(\text{mult}(w,s')-\text{mult}(w,t')\right)^2 : s'\sqsubseteq s,t'\sqsubseteq t,|s'|=|t'|=\lambda\}$.

For a string x, let $\text{mult}_q(x)$ denote the vector of length $|\Sigma|^q$ containing the multiplicities of all q-words in x, for some fixed enumeration of Σ^q. If $|s|=|t|=\lambda$, then $d^2(s,t)$ is the Euclidean distance squared (hence the name) of $\text{mult}_q(s)$ and $\text{mult}_q(t)$. In general, the two sequences s and t need not have the same length. When at least one of the two strings s,t is shorter than λ, then the parameters θ and λ are scaled accordingly.

The measure d^2 has been shown to be well suited for Sanger-style expression clustering (Miller *et al.*, 1999), using $q=6$ as word size. This article and that of (Rao *et al.*, 2010) are the only ones to have examined the use of d^2 with second-generation data.

A closely related dissimilarity measure employed for EST clustering is the q-gram distance (Burkhardt *et al.*, 1999; Ukkonen, 1992), defined as $d_{q-\text{gr}}(s,t)=\sum_{w\in\Sigma^q}|\text{mult}(w,s)-\text{mult}(w,t)|$.

2.2 Filters based on common words

Because edit distance or windowed d^2 is quadratic in the length of the two strings, often filters are used: a similarity relation that the pair is first checked for before $d(s,t)$ is computed.

Our previous tool, WCD, uses the H-filter. It passes two words s,t if they have a certain number of $q-$words in common. For reasons

of efficient implementation, the word count is asymmetric: if a word appears x times in s and y times in t, then we add y to the count.

Definition 2.2 ((H,q)-*Similarity*). Let H,q be positive integers. For $s,t\in\Sigma^*$, we say that t is (H,q)-similar to s if it has at least H occurrences of the q-words that are substrings of s [i.e. if $\sum_{w\sqsubseteq s,|w|=q}\text{mult}(w,t)\geq H$].

Typically, $q\approx 6, H\approx 70$. This similarity relation is asymmetric, as the common substrings of s and t are counted with their multiplicity in t (and not in s). For the right choice of H, the (H,q)-filter is a true, non-heuristic filter for d^2, as stated in the following lemma. The proof can be found in the Supplementary Material.

Lemma 1. Let q,θ and λ be given. Set $H=\lambda-q+1-\theta/2$. If $d_q^2(s,t)\leq\theta$, then s is (H,q)-similar to t, t is (H,q)-similar to s.

This filter has probably good performance. Its computation is linear in the length of s and t; however, every pair has to be inspected separately. Therefore, this filter has $\Theta(n^2m)$ run time.

In an attempt to avoid an all-versus-all comparison, Hazelhurst *et al.* (2008) and Kalyanaraman *et al.* (2002) use variants of the k-word similarity filter:

Definition 2.3 (k-*Word Similarity*). Let k be a positive integer. For $s,t\in\Sigma^*$, we say that s and t are k-word-similar if they have at least one substring of length k in common.

The k-word filter can be implemented efficiently using a suffix array. Its fundamental problem is the choice of k. If k is too large, then many pairs with high similarity will be missed; if k is too small, then a quadratic number of candidate pairs will be found. The heuristic is particularly sensitive to the sequence error rate—as it increases, the largest common word that two sequences must share in order to approximately match becomes smaller.

In our previous work, we found setting $k=27$ was reasonably effective for many datasets. But the approach was fragile: for some datasets, $k=27$ was too big (particularly for shorter sequences) or too small (particularly for sequences with repeats).

We now introduce the KABOOM filter, which is an extension of the k-word filter, and combines its efficient implementation with the good filtering properties of the H-filter.

2.3 The KABOOM filter

The idea behind the (k,α,β)-multiword filter (pronounced KABOOM) is to generalize the k-word approach: rather than requiring that two sequences share a relatively long common word, we require that they share several shorter common words. We claim that this has two advantages (which we substantiate through experimentation later): using multiple words allows us to test for longer regions of high similarity rather than short regions of exact similarity; and this is more likely to be biologically relevant.

While we present the KABOOM filter in the context of using the d^2 dissimilarity measure and the WCD tool, the filter could in principle be used by any clustering algorithm.

The KABOOM filter defines a pair of sequences to be similar if they share a fixed number α of common words (substrings) of a fixed length k, and in addition, the first and the last must occur at least a fixed distance β apart. This last condition is introduced in order to avoid too many overlapping matches. We shall count the substrings with multiplicities, i.e. if s has x occurrences of w and t has y, then we shall add xy to our count. [This count is known as D_2 in the literature (see e.g. Rahmann and Rivals, 2000; Reinert *et al.*, 2009), not to be confused with d^2]. Note that k refers to the word length used in the KABOOM filter, while q is the length of the word used in the computation of d^2: typically $k>q$.

Definition 2.4 ((k,α,β)-*Multiword Similarity*). Let k,α,β be positive integers. For $s,t\in\Sigma^*$, let

$$\text{Common}(s,t)=\{(i,j)\,|\,s_i\ldots s_{i+k-1}=t_j\ldots t_{j+k-1}\}$$

be the set of pairs of occurrences of common k-words of s and t. We say that s and t are (k,α,β)-similar if

(1) $|\text{Common}(s,t)|\geq\alpha$;

(2) $\exists\ (i_1,j_1),(i_2,j_2)\in\text{Common}(s,t)$ s.t. $i_2-i_1\geq\beta$; and

(3) $\exists\ (i'_1,j'_1),(i'_2,j'_2)\in\text{Common}(s,t)$ s.t. $j'_2-j'_1\geq\beta$.

Experiments showed that for Sanger-style data, conservative values of these parameters are $k=16,\alpha=3,\beta=45$ (for d^2-threshold of $\theta=40$), while for 454-type data, conservative values are $k=16$, $\alpha=3$, $\beta=16$ (for $\theta=60$): even though we do not have a theoretical guarantee, with these parameters, the filter produces negligibly few false negatives, i.e. pairs (s,t) which are not (k,α,β)-similar but whose d^2 score is below the threshold θ.

Symmetric and asymmetric implementation: if we drop requirement (3), we get an asymmetric KABOOM-heuristic requiring distance β only in one of the two sequences.

3 ALGORITHM

In this section, we detail our algorithm for finding all pairs of sequences that are (k,α,β)-similar.

The suffix array of a string s is an array of length $|s|$ which lists the indices i according to the lexicographical order of the suffixes starting at position i (Manber and Myers, 1993). By the properties of the suffix array, for any non-empty substring $w\sqsubseteq s$, all occurrences of w are listed contiguously in the suffix array, as all suffixes prefixed by w are contiguous in the lexicographic order.

Let sa be the suffix array of the sequence data. Fix $k>0$. A k-*block* of sa is a maximal subarray of the suffix array where the first k characters of the corresponding entries in the text are equal. We define the *modified suffix array* to be the array sa' such that the indices within each k-block of the suffix array sa are reordered in ascending order, and $invs$ to be the inverse mapping of sa', i.e. $invs[sa'[i]]=i$. The modified suffix array sa' has the following property, whose proof is immediate.

Lemma 2. For each w with $|w|=k$, and every pair of occurrences $i\neq i'$ of w in s, $i<i'\iff invs[i]<invs[i']$.

As a small example, consider a set of seven sequences: *aaa, aacggt, gttaaagt, tcggt, gttat, cgg* and *acggt*. Let $k=3$. The sequences are concatenated together (the @ symbol represents a sequence break character) and the suffix array constructed. Figure 1 gives a detail (lines 15–26) of the suffix array sa (left), the suffix that starts at the corresponding position in the text (centre) and the modified suffix array sa'. The Supplementary Material contains the full table.

aaa@aacggt@gttaaagt@tcggt@gttat@cgg@acggt@			
i	**sa**	**Text from sa[i]**	**sa′**
15	16	agt@tcggt@gttat@cgg@acggt@	16
16	29	at@cgg@acggt@	29
17	32	cgg@acggt@	6
18	37	cggt@	21
19	6	cggt@gttaaagt@t...gt@	32
20	21	cggt@gttat@cgg@acggt@	37
21	34	g@acggt@	34
22	33	gg@acggt@	33
23	38	ggt@	7
24	7	ggt@gttaaagt@t...t@	22
25	22	ggt@gttat@cgg@acggt@	38
26	39	gt@	8

Fig. 1. Detail of example suffix array and modified suffix array ($k=3$).

Each k-block is ruled-off: e.g. rows 17–20 and 23–25 are k-blocks. For clarity, we show all 3-words of the text; however, in the implementation, only those are considered which are substrings of one of the sequences s_i, i.e. those that do not contain `@`.

The k-block in our example from positions 17 to 20 inclusive corresponds to all occurrences of the k-word *cgg*. The array *sa* stores the elements of this k-block, 32, 37, 6 and 21, according to the lexicographic order of the corresponding suffixes, while the column sa' shows the entries within the block sorted.

We create the modified suffix array from the suffix array by scanning through it and re-ordering it, but it could probably be done more efficiently by modifying a suffix array generation algorithm.

3.1 The algorithm

An outline of the modified clustering algorithm is shown below. Detailed pseudo-code can be found in Algorithm 1. The main loop of the algorithm considers each sequence s_i in turn. While processing sequence s_i, we record each sequence s_j containing common k-words with s_i, where $j>i$. For each such sequence s_j, we compute count$[j]:=|\text{Common}(s_i,s_j)|$, as well as the leftmost and rightmost occurrences of such words in s_i and in s_j. The variables `lindI`$[j]$ and `rindI`$[j]$ (`lindJ`$[j]$ and `rindJ`$[j]$) store the current leftmost and rightmost positions in s_i (s_j) of all common k-word of s_i and s_j.

for all sequences s_i **do**
  $M \leftarrow \emptyset$
  for all words $w \sqsubseteq$ sequence s_i **do**
    $M \leftarrow M \cup \{j : j>i, w \sqsubseteq \text{sequence } s_j\}$
  for all $j \in M$ where `count`$(i,j) \geq \alpha \wedge$ `rindI`$[j]-$`lindI`$[j] \geq \beta \wedge$ `rindJ`$[j]-$`lindJ`$[j] \geq \beta$ **do**
    if sim(s_i,s_j) **then**
      merge(`cluster`(i), `cluster`(j))

In more detail: when processing sequence s_i, we consider each (overlapping) k-word in s_i. Let w be a k-word in sequence s_i that starts at position r. Using the inverse of the modified suffix array *invs*, we find where this word occurs in sa', say at position $p=invs[r]$. Then we find all occurrences of w in other sequences s_j, where $j>i$, by looking at each entry $p'>p$ within the same k-block. By Lemma 2, this gives all occurrences to the right of the current occurrence of w. With this information, we update our records of which sequences s_j share a k-word with s_i, and also where, in both s_i and s_j, the leftmost and the rightmost of these common k-words are.

Once all k-words in sequence s_i have been processed, we know which sequences s_j, for $j>i$, share a k-word with s_i, how many k-words are shared and the various leftmost and rightmost positions. Sequence s_i is a candidate sequence with each of those sequences satisfying the condition of the KABOOM filter. A small example is given in the Supplementary Material.

WCD-EXPRESS then compares each candidate pair first with the heuristics described by Hazelhurst (2008) and then, if necessary, using d^2. However, any dissimilarity function (such as edit distance) could be used. Note that every pair (s_i,s_j), $i<j$, is considered only once, namely in the iteration for s_i.

Algorithm 1 Algorithm WCD-EXPRESS (with KABOOM)

for $i \leftarrow 0$ to $n-1$ **do**
  $r \leftarrow 0$; $M \leftarrow \emptyset$; {r: index in string s_i}
  while $r \leq$ `length`$(s_i)-k$ **do**
    $p \leftarrow$ `invs`$[r]+1$; {p: index in sa'}
    while (`seqnumber`[`sa'`$[p]]=i \wedge$ `not`(`newblock`$[p]$)) **do**
      p++; {skip further matches within s_i}
    while `not`(`newblock`$[p]$) **do**
      $j \leftarrow$ `seqnumber`[`sa'`$[p]]$; {j: current sequence}
      if $j \notin M$ **then**
        `lindI`$[j] \leftarrow r$; `rindI`$[j] \leftarrow r$; {first common word between s_i and s_j seen}
        $M \leftarrow M \cup \{j\}$;
      else
        `rindI`$[j] \leftarrow r$;
      `lindJ`$[j] \leftarrow \min($`sa'`$[p]$, `lindJ`$[j])$;
      `rindJ`$[j] \leftarrow \max($`sa'`$[p]$, `rindJ`$[j])$;
      `count`$[j]$++; {`count`: # common words of s_i and s_j}
      p++;
    r++;
  {compute $d^2(s_i,s_j)$ for j passing filtering phase}
  for all $j \in M$ **do**
    if (i,j) are (k,α,β) similar **then**
      if $d^2(s_i,s_j) \leq \theta$ {distance of s_i and s_j is small} **then**
        merge(`cluster`(i), `cluster`(j));
  reset `lindI`, `rindI`, `lindJ`, `rindJ`, `count`, M;

3.2 Analysis of algorithm

Suppose there are n sequences of average length m in set S. For Sanger-style sequencing m may be between 300 and 700. For real datasets n is unlikely to be $<10^4$ and may be as large as 10^6 (so $m<n$, and in large datasets $m^2<n$). Let k be the word length used in the suffix array algorithm step. Typical values of k will be in the range 12–20. We make the following definitions:

c : number of candidate pairs found by WCD.
c_K: no. of candidate pairs found by the KABOOM filter.
c_X: number of candidate pairs found by WCD-EXPRESS, (using the KABOOM filter and all other heuristics).
p : number of distinct k-words occurring in S.
η_w: number of occurrences of the k-word w in S.

γ: $\frac{p}{mn}$, ratio of number distinct words to total number of k-words. ($1/\gamma$: average no. occurrences of k-words.)
E: $\frac{1}{p}\sum_{|w|=k}\eta_w^2$, ave. of squares of no. of occurrences.

The previous version of WCD needs $\Theta(mn^2)$ time for the heuristics and $\Theta(cm^2)$ time for computing the d^2 or edit distance of those pairs of sequences that have been passed by the heuristics. In WCD-EXPRESS, in the KABOOM-step, for each word w in the file, we do $O(\eta_w)$ work. Each distinct word w appears η_w times, so the total work done is $O(\sum_{|w|=k}\eta_w{}^2)=O(\gamma Emn)$. Another way of looking at this is that there are mn (non-distinct) words, and for each we do γE work on average.

WCD-EXPRESS then applies all of WCD's heuristics on the pairs passed, which run in linear time. Thus, the total amount of work done by WCD-EXPRESS is $\Theta(\gamma Emn+c_K m+c_X m^2)$. Since WCD-EXPRESS applies all of WCD's heuristics, $c_X \leq c$. In practice, $c \in O(n)$. Thus, provided $\gamma E < n$, WCD-EXPRESS will run faster than WCD. However, since the constant factors due to memory behaviour are likely to be substantially larger, for practical purposes, it is important that $\gamma E \ll n$. The experimental results reported later demonstrate that in practice WCD-EXPRESS runs substantially faster than WCD and that indeed $\gamma E \ll n$.

For the generation of the suffix array, any of a large number of suffix array construction algorithms can be used, several of which run in linear time. In our current implementation, we use the *mkESA* tool (Homann *et al.*, 2009), which employs the Deep-Shallow algorithm (Manzini and Ferragina, 2004), one of several super-linear algorithms that have been shown to perform better in practice. See (Puglisi *et al.*, 2007) for a survey of suffix array construction algorithms.

WCD-EXPRESS requires substantial amount of RAM to store both the suffix array and its inverse: $\Theta(mn\log mn)$ amount of RAM.

4 IMPLEMENTATION

The algorithm is implemented as an extension to the WCD clustering tool (Hazelhurst *et al.*, 2008). The code is implemented in C.

Pre-processing: pre-processing requires building a suffix array of the data file and its reverse complement. We use the *mkESA* tool (Homann *et al.*, 2009) to create the suffix array, which suited our needs well and performed excellently.

Clustering: the WCD-EXPRESS program performs clustering as presented in Algorithm 1. However, it is important to note that the call to the d^2 algorithm is preceded by the use of filtering heuristics described in Hazelhurst (2008); Hazelhurst *et al.* (2008).

5 RESULTS

This section compares WCD-EXPRESS to the previous version of WCD, and to two other systems, TGICL (Pertea *et al.*, 2003) and PEACE (Rao *et al.*, 2010), evaluating the impact of the KABOOM heuristic and the overall performance of WCD-EXPRESS.

All experiments were done on an Intel E5335 (2 GHz; dual quad-core processor with 4 MB of L2 cache per processor and 16 GB of RAM; single thread; Scientific Linux 5.4, `gcc` 4.1.2, O-2 for WCD-EXPRESS and WCD). We used the asymmetric implementation of WCD-EXPRESS, which initial experimentation showed was slightly faster (but which is controlled by a compiler-switch).

Table 1. Comparison of new and old versions of WCD

Dataset	# seqs (k)	Size MB	γE	WCD-EXP (s)	WCD (s)	Speed-up
A076941	77	32	17	100	578	5.7
A208	484	208	102	1240	23983	19.3
C10	126	56	355	511	4512	8.8
chlamy	190	100	139	1000	5989	6.0
Drosophila	25	86	68	184	1542	8.4
ecHuman	17	11	171	135	496	3.7
pubcot	30	17	34	65	222	3.4
ricinus	58	40	162	840	1518	1.8
xen	233	137	63	855	9298	10.8

Experimental results on different sets of EST datasets. # seqs is the number of sequences in thousands; *size* is the number of megabases. E is the average square of the frequency of the distinct words; γ is the ratio of the number of distinct words to the total number of words. WCD-EXPRESS is the time our new algorithm takes including pre-processing; WCD is the time our previous version of WCD takes. All times in seconds are rounded to the nearest second. For both versions of WCD, the same parameters were used. The sensitivity of WCD-EXPRESS with respect to WCD is over 0.999 in all cases.

5.1 Data

The experiments use both real and synthetic Sanger-style and second-generation data. The data are described in the Supplementary Material. In summary, A686904 is a set of 686904 *Arabidopsis thaliana* sequences from GenBank, while other datasets of the form Ax are subsets of this; ecHuman is the EasyCluster reference set (Picardi *et al.*, 2009); chlamy a set of *Chlamydomonas reinhardtii* sequences used to test PEACE (Rao *et al.*, 2010). The C-series is a set of synthetic data files, generated using the ESTsim tool (Hazelhurst and Bergheim, 2003); the metasim files are synthetic files simulating 454 and Illumina style sequences using the tool MetaSim (Richter *et al.*, 2008).

5.2 Impact of the KABOOM filter

Table 1 shows the improvement in running time from WCD to WCD-EXPRESS, gained by adding the KABOOM filter. We use a number of Sanger-style datasets using the same parameters for both tools. The running times for WCD-EXPRESS include the generation of the suffix array. In all cases, there is a large speed-up, and the speed-up is larger with larger datasets. Recall from Section 3.2 that the success of the algorithm depends on how γE compares to the number of sequences. As can be seen from Table 1, on all our datasets γE is two to three orders of magnitude smaller than n.

Next, we explore the impact of the KABOOM filter with different word sizes on the quality of the clustering and running times. Figure 2 shows the time taken and quality of the clustering of different datasets as the suffix word length k changes, on different data files (both real and synthetic). H is the parameter of the (H,q)-filter (Section 2.2). For quality, we measure sensitivity of the clustering with respect to a base case of suffix array word length of 10 (a very conservative value). The other clustering parameters are chosen extremely conservatively to avoid masking too aggressive values of other parameters. These results show that choosing a suffix word length of 16 does not significantly decrease sensitivity but that for word length at least 12 the running time improves dramatically.

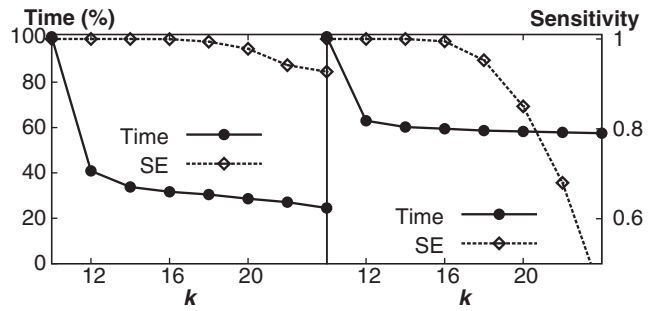

Fig. 2. The effect of word size on the quality and time. The graph on the left shows a set of mouse (Sanger) data and the graph on the right shows the Metasim 454 data. The left *y*-axis shows time (as a percentage of the time taken when $k = 10$), and the right *y*-axis shows sensitivity.

5.3 Quality comparison with other tools

Given a clustering $\mathcal{C}$ of a set S, quality is measured using validity indices, which compare $\mathcal{C}$ to a known clustering $\mathcal{D}$. These give a numerical value in the range $[0, 1]$ (1 being best value), in terms of the number of pairs (x, y) which are put into the same cluster by both $\mathcal{C}$ and $\mathcal{D}$ [true positives (TP)]; those which $\mathcal{C}$ clusters together but $\mathcal{D}$ does not [false positives (FP)]; those which neither clusters together [true negatives (TN)]; and those which $\mathcal{D}$ clusters together but $\mathcal{C}$ does not [false negatives (FN)]. Common indices include *sensitivity* $\mathrm{SE} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$, *positive predictive value* $\mathrm{PPV} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$ and the *Jaccard index* $\mathrm{JI} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN} + \mathrm{FP})$.

Table 2 compares the running times and quality of WCD-EXPRESS and PEACE on several datasets with known clusterings (the first three are Sanger-style data, the others second-generation). WCD-EXPRESS and PEACE have very similar quality scores, and WCD-EXPRESS consistently outperforms PEACE with respect to run-time.

A complication in comparison is that PEACE filters out low-complexity sequences in a pre-processing step, while the WCD tools do not. Therefore, for Table 2 we adopt a pre-step before calling WCD-EXPRESS of filtering out the same sequences that PEACE does. Note that our results differ from those quoted in Rao *et al.* (2010) because we adopt a different methodology for dealing with the filtered sequences. See the Supplementary Material for a full discussion. Since TGICL incorporates filtering into its clustering step in a different way, a direct comparison with WCD-EXPRESS was not possible. However, we note that (Rao *et al.*, 2010) report a comparison of TGICL with PEACE and with WCD and found the clustering quality to be competitive.

The clusterings computed by WCD-EXPRESS and WCD are essentially the same. Note that the scores for 454 and Illumina data in Table 2 differ from those reported in (Rao *et al.*, 2010). The reason is that the default parameters of WCD were optimised for Sanger-style data, while for short read sequences, other values are appropriate. (For details see Supplementary Material.)

As shown in our earlier work, the choice of parameters has a very important effect on the quality of the results (Zimmermann *et al.*, 2004). In most published work, tools are compared based on one set of parameters. This kind of study is limited, since it only proves that with one set of parameters one tool performs better or worse than another tool with its own set of parameters.

Table 2. Quality and runtime comparison of WCD-EXPRESS (column labelled WCD) and PEACE on datasets where an ideal clustering is known

	Sensitivity		Jaccard Index		Time (s)	
	WCD	PEACE	WCD	PEACE	WCD	PEACE
A076941	0.932	0.930	0.473	0.477	100	951
chlamy	0.949	0.949	0.513	0.513	907	8823
ecHuman	0.996	0.998	0.707	0.630	50	147
metasim454	0.793	0.714	0.765	0.689	16	66
metasimIllum	0.444	0.368	0.398	0.364	19	1975

The first three datasets are Sanger style data (average length 500); the second two are 454 (average length 240) and Illumina data (average length 60). See text and Supplementary Material for details.

We stress that it is unlikely that any tool has a set of universally optimal parameters. The range of different sequencing technologies and quality of data means that different modelling parameters will be required to cluster optimally. This makes computational performance more important. A fast tool is very helpful since it gives an experimenter the ability to cluster the same dataset with different parameters and investigate the stability of the clustering with respect to the various parameters.

We also ran tests that showed that for reasonable values of k, very high levels of sensitivity can be obtained. For Sanger-style data with $k = 16$, even for an aggressive $\theta = 60$, WCD-EXPRESS gives less than a 0.1% FN rate. For 454 data, with $k = 14$, and aggressive clustering values we get less than a 0.5% FN rate. As expected, performance declines with shorter sequence length; however, second-generation sequence length is increasing as the technology improves.

5.4 Computational cost comparisons with other tools

Memory usage: the use of the suffix array and inverse creates significant memory requirements for WCD-EXPRESS. The current implementation requires ∼25 bytes of RAM per byte of input so that a 200 MB input file requires ∼5 GB of RAM to run effectively. This makes WCD-EXPRESS much more expensive than WCD (which required ∼0.3 bytes of RAM per byte of input).

TGICL's memory footprint is very small as the input file is broken into chunks. PEACE's memory usage pattern varies. PEACE's usage is 2–10 times less than WCD-EXPRESS's, with the difference diminishing on larger data (2.1 GB for the 200 MB input file mentioned above). A positive feature of WCD-EXPRESS's memory usage is that memory is allocated as the data are read in—thus even on a long run, usage will be known after a minute or so. PEACE's memory usage, on the other hand, may increase throughout its execution.

The Supplementary Material presents experimental evidence and discusses memory use in more detail.

Running times: the tables and figure below show computational costs of WCD-EXPRESS, PEACE and TGICL on a variety of data. As discussed, TGICL and PEACE do filtering while WCD-EXPRESS does not. The Supplementary Material shows that this has a profound effect on quality and performance. If tool *A* filters and *B* does not, it may appear that *A* is faster than *B*, or *vice-versa* when the root cause is that the data to be clustered are effectively very different. In the experiments below, we adopt a filtering pre-step for WCD-EXPRESS.

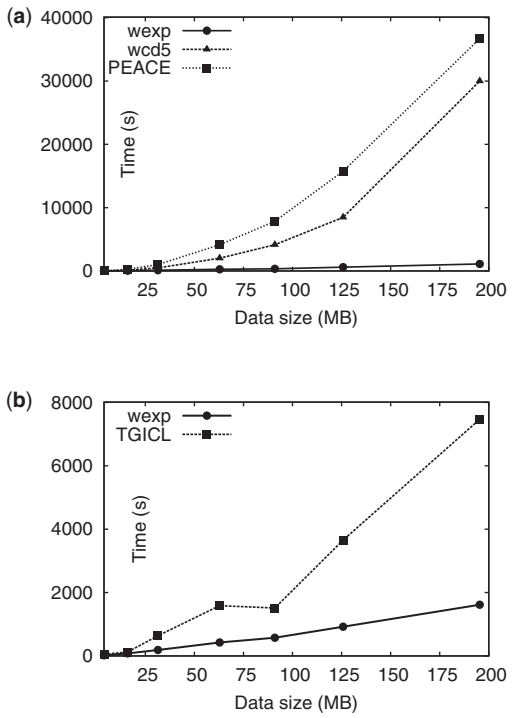

Fig. 3. Performance on the Arabidopsis dataset series on an Intel E5335 Xeon processor. We compare WCD-EXPRESS with PEACE and TGICL on separate graphs to make the differences clearer and because different filtering is done by TGICL and PEACE. Note the scales on the *y*-axes differs. The slight jink in TGICL's performance curve was re-tested several times. (**a**) PEACE versus WCD and WCD-EXPRESS. (**b**) TGICL versus WCD-EXPRESS.

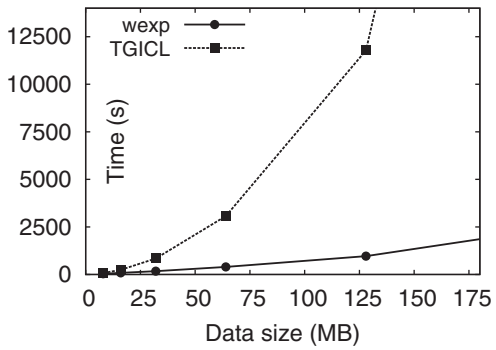

Fig. 4. Comparison of performance WCD-EXPRESS versus TGICL on subsets of a large set of Human 454 ESTs.

To be fair, we do this differently in the comparisons with TGICL and PEACE, as these tools filter differently (see Supplementary Material for details). We emphasize that the times reported for WCD-EXPRESS take into account all pre-processing time, including filtering and construction of the suffix array.

In Figure 3, we show how the running times scale as datasets grow larger. The *Arabidopsis* data is a set of real ESTs. Figure 4 shows the difference between WCD-EXPRESS and TGICL on subsets of Human 454 ESTs of different sizes. See the Supplementary Material for additional experimentation.

Table 3. Running times on different datasets (top part Sanger-style, bottom second-generation)

Dataset	No. of sequences (K)	Size (Mb)	Time (s)			
			WCD	PEACE	WCD	TGICL
A076941	77	32	100	951	166	282
A208	484	208	1078	36611	1620	7111
C10	126	56	135	1800	278	448
chlamy	190	100	907	8823	900	4577
droste	83	45	141	1130	247	431
ecHuman	17	11	50	147	115	233
pubcot	30	17	43	124	85	150
ricinus	58	40	783	923	555	1085
xen	233	137	496	7032	1065	2218
metasim454	25	6	16	66	25	46
hsub128	355	130	1023	25706	964	11788
metaIllum	150	9	19	1975	25	121
soybean	882	173	1626	167590	4012	39556
SRR019551	335	87	876	21595	3204	63524

All times rounded to the nearest second. See text for details.

Table 3 shows performance of WCD-EXPRESS on a range of different datasets. In summary, the results show that WCD-EXPRESS outperforms PEACE, and for very large datasets does so substantially. WCD-EXPRESS is also much faster than TGICL. Additional experimentation and more detailed results (including pre-processing costs) are presented in the Supplementary Material.

6 CONCLUSION AND FUTURE RESEARCH

This article has introduced a new algorithm for finding candidate pairs for clustering gene expression data. The idea is that two sequences are candidates for comparison if they share α many common k-words, where the leftmost and rightmost words start at least β away from each other. This heuristic can be implemented very efficiently using a modified suffix array and its inverse.

The experimental results show that the algorithm is very effective. For reasonable values of word length, substantial improvement in compute performance is achieved without degradation in quality of clustering. There are a number of areas for future work:

Improvement of the algorithm implementation: there is scope for improving the current implementation, using less memory and improving cache behaviour. We currently have an experimental version of our tool that uses about half the amount of memory, with an approximate 15% run-time penalty.

Parallelization: at this point, WCD-EXPRESS is not parallelized (WCD has both pthreads and MPI parallelization). Parallelization should be straightforward, though given the overall cost of pre-processing, for large-scale parallelization, suffix array construction and sequence filtering must both be parallelized. A number of implementations of parallel suffix array construction are available, including in *mkESA*, which can be used directly.

Improving clustering quality: the experiments show that the quality of clustering is very dependent on parameters used. The most important lesson is that in practice, bioinformatists should run their

tools several times with different parameters to evaluate the stability of their results. Work is needed to separate out the effect of the parameters used and the underlying algorithms. This is also likely to be affected by the error models of the underlying sequencing technology (e.g. the homopolymer problem for 454 data).

ACKNOWLEDGEMENTS

We thank F. Cicalese for his help and support, the referees for helpful comments, and Dr D.J. Rao for help with PEACE.

Funding: Anderson-Capelli Fund; South African National Research Foundation (grant number IFR2010052500011); SA Centre for High Performance Computing (to S.H.); European Union's Seventh Framework Programme (FP7/2007-2013); Marie Curie Intra-European Fellowship for Career Development (PIEF-GA-2010-274778 to Zs.L.).

Conflict of Interest: none declared.

REFERENCES

Burkhardt,S. *et al.* (1999) *q*-gram based database searching using a suffix array (QUASAR). In *Proceedings of the Third Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, ACM, New York, pp. 77–83.

Hazelhurst,S. (2008) Algorithms for clustering EST sequences: the wcd tool. *South African Comput. J.*, **24**, 1542–1546.

Hazelhurst,S. and Bergheim,A. (2003) ESTsim: a tool for creating benchmarks for EST clustering algorithms. *Technical Report TR-Wits-CS-2003-1*. School of Computer Science, University of the Witwatersrand, Johannesburg, S.A.

Hazelhurst,S. *et al.* (2008) An overview of the wcd EST clustering tool. *Bioinformatics*, **24**, 1542–1546.

Homann,R. *et al.* (2009) *mkESA*: enhanced suffix array construction tool. *Bioinformatics*, **25**, 1084–1085.

Huang,X. and Madan,A. (1999) CAP3: a DNA sequence assembly program. *Genome Res.*, **9**, 868–877.

Jain,A.K. *et al.* (1999) Data clustering: a review. *ACM Comput. Surv.*, **31**, 264–323.

Kalyanaraman,A. *et al.* (2002) Parallel EST clustering. In *Proceedings of IEEE Conference High Performance Computational Biology*. IEEE Computer Society.

Levenshtein,V. (1966) Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Phys. Doklady*, **10**, 707–710.

Malde,K. *et al.* (2003) Fast sequence clustering using a suffix array algorithm. *Bioinformatics*, **19**, 1221–1226.

Manber,U. and Myers,E.W. (1993) Suffix arrays: a new method for on-line string searches. *SIAM J. Comput.*, **22**, 935–948.

Manzini,G. and Ferragina,P. (2004) Engineering a lightweight suffix array construction algorithm. *Algorithmica*, **40**, 33–50.

Miller,R. *et al.* (1999) A comprehensive approach to clustering of expressed human gene sequence: the sequence tag alignment and consensus knowledge base. *Genome Res.*, **9**, 1143–1155.

Pertea,G. *et al.* (2003) TIGR gene indices clustering tools (TGICL): a software system for fast clustering of large EST datasets. *Bioinformatics*, **19**, 651–652.

Picardi,E. *et al.* (2009) EasyCluster: a fast and efficient gene-oriented clustering tool for large-scale transcriptome data. *BMC Bioinformatics*, **10** (Suppl. 6), S10.

Pop,M. and Salzberg,S. (2008) Bioinformatics challenges of new sequencing technology. *Trends Genetics*, **24**, 142–149.

Puglisi,S.J. *et al.* (2007) A taxonomy of suffix array construction algorithms. *ACM Comput. Surv.*, **39**, 1–31.

Rahmann,S. and Rivals,E. (2000) Exact and efficient computation of the expected number of missing and common words in random texts. In *Proceedings of the 11th Annual Symposium Combinatorial Pattern Matching (CPM 2000)*, Vol. 1848 of *Lecture Notes in Computer Science*, Springer, pp. 375–387.

Rao,D. *et al.* (2010) PEACE: parallel environment for assembly and clustering of gene expression. *Nucleic Acids Res.*, **38**, W737–W742.

Reinert,G. *et al.* (2009) Alignment-free sequence comparison (I): Statistics and power. *J. Comput. Biol.*, **16**, 1615–1634.

Richter,D. *et al.* (2008) MetaSim – a sequencing simulator for genomics and metagenomics. *PLoS One*, **3**, e3373.

Robison,K. (2010) Editorial: next generation sequencing. *Brief. Bioinformatics*, **11**, 455–456.

Schwartz,D.C. and Waterman,M.S. (2009) New generations: Sequencing machines and their computational challenges. *J. Comput. Sci. Technol.*, **25**, 3–9.

Slater,G. (2000) *Algorithms for the Analysis of Expressed Sequence Tags*. PhD Thesis, University of Cambridge, Cambridge, UK.

Torney,D. *et al.* (1990) Computation of d^2: a measure of sequence dissimilarity. In Bell,G. and Marr,T. (eds) *Computers and DNA*. Addison-Wesley, Boston, Mass, pp. 109–125.

Ukkonen,E. (1992) Approximate string-matching with *q*-grams and maximal matches. *Theor. Comput. Sci.*, **92**, 191–211.

Zimmermann,J. *et al.* (2004) A method for evaluating the quality of string dissimilarity measures and clustering algorithms for EST clustering. In *Proceedings of the 4th IEEE International Symposium BioInformatics and BioEngineering (BIBE 2004)*. IEEE Computer Society, pp. 301–309.