Compact universal *k*-mer hitting sets

Yaron Orenstein[1], David Pellow[2], Guillaume Marçais[3], Ron Shamir[2], and Carl Kingsford[3]

[1] Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA
[2] Blavatnik School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel
[3] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
yaronore@mit.edu, dpellow@tau.ac.il, gmarcais@cs.cmu.edu, rshamir@tau.ac.il, carlk@cs.cmu.edu

1 Background

We consider the following problem involving covering strings by selecting short k-mer substrings:

Problem 1. Given integers k and L, find a smallest set U_{kL} of k-mers such that any string of length L or longer must contain at least one k-mer from U_{kL}.

The set U_{kL} is called a *universal* set of hitting k-mers, and we call each k-mer in the set *universal*. Such a set has a number of applications in speeding up genomic analyses since it can often be used in places where minimizers have been used in the past: hashing for read overlapping, sparse suffix arrays and Bloom filters to speed up sequence search.

A universal set U_{kL} has a number of advantages over minimizers for these applications. First, the set of minimizers for a given collection of reads may be as dense as the complete set of k-mers, whereas we show below that U_{kL} is often smaller by a factor of k. Second, for any k and L, the set of universal k-mers needs to be computed only once and not recomputed for every dataset. Third, the hash buckets, sparse suffix arrays, and Bloom filters created for different datasets will contain a comparable set of k-mers if they are sampled according to U_{kL}. The universal set of k-mers also has the advantage over dataset-specific sets because one does not need to look at all the reads before deciding on the k-mers to use, and one does not need to build a dataset-specific de Bruijn graph to select covering k-mers.

The problem is also of theoretical interest as it can be rephrased as an equivalent problem on the complete (original) de Bruijn graph:

Problem 2. Given a de Bruijn graph D_k of order k and an integer L, find a smallest set of vertices U_{kL} such that any path in D_k of length $L-k$ passes through at least one vertex of U_{kL}.

We show that the problem of finding a minimum-size k-mer set that hits every string in a given set of L-long strings is NP-hard, further motivating the need for a universal k-mer set. We provide a heuristic called DOCKS that is based on the combination of three ideas. First, we use a decycling algorithm due to Mykkeltveit to convert a complete de Bruijn graph into a directed acyclic graph (DAG) by removing a minimum number of k-mers. We then supply a novel dynamic program to score remaining k-mers by the number of remaining length-ℓ paths that they hit. Finally, we use that dynamic program in a greedy heuristic to select the additional k-mers and produce a small universal set $\hat{U}_{kL}$, which we show empirically to often be close to the optimal size. Our use of a greedy heuristic is motivated by providing a proof that finding a small ℓ-path cover in a graph G is NP-hard even when G is a DAG. We report on the size of the universal k-mer hitting set produced by DOCKS and demonstrate on two datasets that we can better cover sequences with a smaller set of k-mers than is possible using minimizers. Our results also provide a starting point for additional theoretical investigation of these path coverings of de Bruijn graphs.

2 Results

2.1 DOCKS algorithm

To get the algorithm, we combine the two steps. First, we find a minimum-size decycling set in a complete de Bruijn graph of order k and remove it from the graph, turning it into a DAG. Then, we repeatedly remove a vertex v with the largest hitting number $T_\ell(v)$ (the number of ℓ-long paths the vertex participates in) until there are no ℓ-long paths, where $\ell = L-k$, recomputing $T_\ell(u)$ for all remaining u after each removal. This hitting number can be computed efficiently using dynamic programming. This is summarized below (Algorithm DOCKS).

The running time is polynomial in L and $|\Sigma|^k$. Finding the decycling set takes $O(|\Sigma|^k)$, as the size of the set is $\Theta(|\Sigma|^k/k)$ and the running time for finding each k-mer is $O(k)$. In the second phase, each iteration calculates the hitting number of all vertices using dynamic programming in time $O(|\Sigma|^k L)$. The number of iterations is $1+p$, the number of vertices removed. Thus, the total running time is dominated by steps 4–8 and is $O((1+p)|\Sigma|^k L)$.

Algorithm 1 DOCKS: Find a small k-mer set hitting all L-long sequences

1: Generate a complete de Bruijn graph G of order k, set $\ell = L - k$.
2: Find a decycling vertex set X using Mykkeltveit's algorithm.
3: Remove all vertices in X from graph G, resulting in G'.
4: **while** there are still paths of length ℓ **do**
5: Calculate the number of starting and ending i-long paths at each vertex, for $0 \leq i \leq \ell$.
6: Calculate the hitting number for each vertex.
7: Remove a vertex with maximum hitting number from G', and add it to set X.
8: **end while**
9: Output set X.

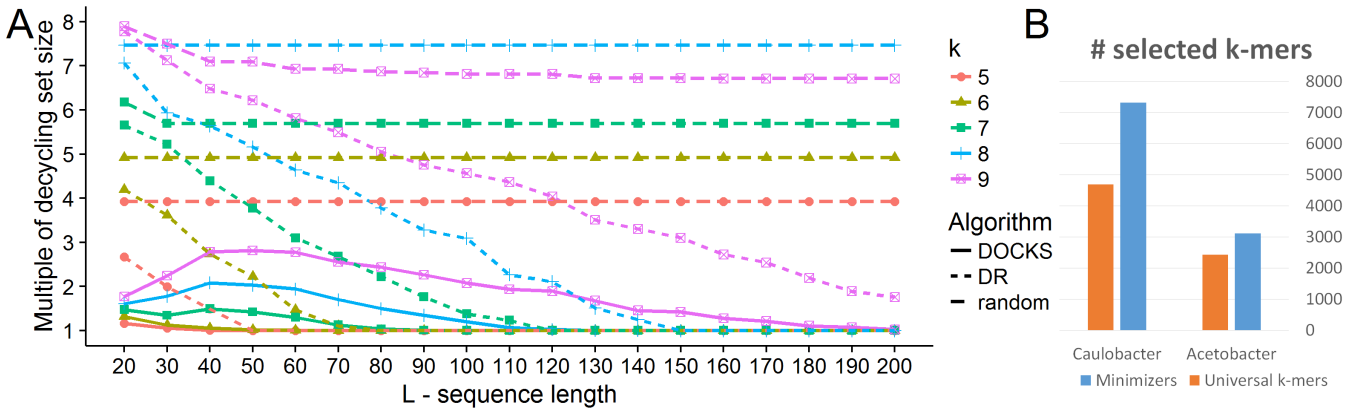

Fig. 1. Performance of DOCKS. A) For different combinations of k and L we ran DOCKS and two random procedures over the DNA alphabet. The results are shown in comparison to the size of the decycling set. When the ratio is 1, all the sequences avoiding the decycling set were of length shorter than L. DR: decycling+random. B) Comparison of the number of selected minimizers and universal k-mers, for $k = 8, L = 100$ in bacterial genomes.

2.2 Computational results

We implemented and ran DOCKS over a range of k and L: $5 \leq k \leq 9$ with $20 \leq L \leq 200$, in increments of 10. These are typical values used for minimizers of longer k-mers and read lengths of short read sequences. We also implemented two random procedures that we compare to as baselines. One, termed "random", removes random vertices until no $\ell = L - k$ paths remains. The second, termed "decycling+random" (DR), first removes a minimum-size decycling set and then randomly removes vertices until no path of length $\ell = L - k$ exists. The results are summarized in Figure 1A. Our method outputs a set of k-mers that is much smaller than both random procedures.

In Figure 1B, we compare the size of the universal hitting k-mers and the minimizers in two bacterial genomes. *Acetobacter tropicalis* (RefSeq NZ_CP011120) has a genome of 2.8 Mbp and a GC content of 47.8%. *Caulobacter vibriodes* (RefSeq NC_002696) is larger at 4.0 Mbp and has a higher GC content of 67.2%. For each genome, we computed the number of minimizers using $k = 8$ and a window length of 100. Also, for each window of 100 bases we found a k-mer from the set $\hat{U_{kL}}$ for $k = 8, L = 100$, computed by DOCKS. Each such window is guaranteed to contain at least one universal k-mer, and usually more than one. In each window, we select only one of the universal k-mers, the smallest one in lexicographic order. Using universal hitting k-mers instead of minimizers gives a smaller set of selected k-mers.

3 Conclusion

In this work, we presented the DOCKS algorithm, which generates a compact set of k-mers that together hit all L-long DNA sequences. DOCKS's good performance can be attributed to its two components. It first optimally removes a minimum-size set that hits all infinite sequences, which takes care of most L-long sequences. It then greedily removes vertices that hit remaining L-long sequences. Its feasibility stems from the first step, which runs in time $O(k)$ times the size of the output, and the second step, which uses dynamic programming to bound the running time to be quadratic in the output size times L.

We demonstrated the ability of DOCKS to generate compact sets of k-mers that hit all L-long sequences. These k-mer sets can be generated once for any desired value of k and L and then used easily for many different purposes. For example, there is a set of only 700 6-mers out of a total of 4096 that hits every sequence longer than 70 bases — a typical read length for many sequencing experiments — enabling efficient binning of reads. These sets of k-mers could improve many of the applications that use minimizers, as we showed that they are both smaller and more evenly distributed across typical sequences.

DOCKS provides the first practical solution to the identification of universal sets of k-mers. The software is freely available on `acgt.cs.tau.ac.il/docks/`, as are universal sets of k-mers over a range of values of L and k.

This work is under review at WABI 2016.