We describe two algorithms that take advantage of the long range "independence" in DNA strings. The first algorithm, coupled-seeds, extends gapped seeds to allow indels. The second algorithm is a pre-alignment filter which produces relatively small lists of candidate alignments.
The two algorithms are independent, but they can be combined. Coupled-seeds are compatible with hash-table based alignment algorithms. Pre-alignment filters can be combined with existing approaches to alignment; the filters produce small lists of candidates which allow any alignment algorithm to rapidly dismiss many candidates that would have otherwise been processed.
We simulate reads from a repetitive human reference genome to demonstrate the use of pre-alignment filters in alignment of paired-end reads, and the use of pre-alignment filters with coupled-seeds in alignment of long reads with high indel rates (PacBio-like reads).

# Using the Long Range "Independence" in DNA: Coupled-Seeds and Pre-Alignment Filters.

Roy Lederman[†]
Technical Report YALEU/DCS/TR-1477
August 7, 2013

[‡] Applied Mathematics Program, Yale University, New Haven CT 06511

**Keywords:** *DNA, Alignment, Mapping, Pre-Alignment, Seeds.*

## 1 Introduction

Read alignment is a common step in the sequencing of DNA. Various algorithms have been designed for read alignment. The two main classes of algorithms are a) prefix-tree algorithms and b) hash-table based algorithm. Hash-table algorithms can be subdivided into seed-and-extend and q-gram/"dot plot" algorithms. Seed-and-extend algorithms can be further subdivided into continuous-seed and gapped-seed algorithms. We proposed an additional class of randomized permutations-based algorithms in [1].

One of the common problems in read alignment is that DNA is often highly repetitive, so there are often many possible places in which substrings of a read could be approximately aligned. Indeed, the entire read can often be approximately aligned in several different locations in the reference. However, when one examines longer reads and longer substrings of DNA, they appear less repetitive, or more "independent." These properties have been observed and used in several algorithms such as PatternHunter[2]. We discuss one of the ways to measure this property in [3].

Unfortunately, many alignment algorithms have some form of restriction that makes them consider short substrings first. Prefix-tree algorithms examine contiguous substrings and seed-and-extend algorithms use short contiguous seeds. Gapped-seed-and-extend and basic permutations-based algorithms, can use long substrings, but they require substrings without indels. As a result, gapped-seed-and-extend algorithms and permutations-based algorithms are also restricted to short substring when many indels are allowed. q-gram/"dot plot" algorithms use very short q-grams, and they create "dot-plots" that are less vulnerable to indels. However, q-gram algorithms require more processing and can become slower when the references are large and when the reads are allowed to have many errors.

In this manuscript we propose two separate algorithms that take advantage of the long range "independence" in DNA.

The first algorithm replaces standard seeds with "coupled-seeds" which use long substrings, but are less vulnerable to indels.

The second algorithm is a "pre-alignment-filter" which uses the long range "independence" to produce a small list of candidates. For many of the reads, this filter produces only one candidate, the correct one. In large repetitive references, and short reads, some of the reads are likely to have multiple candidates. The idea is to pass this list of candidates to special implementations of any other existing alignment algorithm, allowing that existing algorithm to rapidly eliminate many of the candidates that would have otherwise been considered.

## 2 Preliminaries

### 2.1 Seeds and q-gram algorithms

Many search algorithms use “seeds” and “q-grams.” In this section, we present a general, simplified description of such algorithms.

In their simplest form, seeds-based algorithms use k-mers as seeds. k-mers are contiguous substrings of length $k$. Gapped versions of these algorithms extend the algorithms to non-contiguous substrings.

A “seed and extend” algorithm, like Blast [4] or mrFast [5] uses a list of the occurrences of k-mers in the reference. When a read is examined, the algorithm chooses several k-mers from the read, and creates a list of all the occurrences of that k-mer in the reference. The algorithm then attempts to compare the read to the regions where the k-mers are found in the reference.

A q-gram/“dot plot” algorithm like RazerS [6] usually treats the reference in small segments. For each segment and each read, it builds a “dot plot” of matching q-grams. These “dot plots” can be used to create sketch alignment of reads to the reference.

The pigeon-hole principle and the q-gram lemma [7] provide some of the principles for the analysis of these approaches. Several algorithms have been proposed for counting occurrences of seeds in strings and of q-grams in “dot plots” [8, 9, 10] in order to find candidate matches.

### 2.2 Short range dependency and long range dependency

It is known that in DNA, seeds of $k$ consecutive characters are more ambiguous than gapped seeds of $k$ non-consecutive characters [2]. In other words, a seed of $k$ non consecutive characters is likely to produce fewer candidates than a seed of $k$ consecutive characters. The longer the substring from which the characters are sampled, and the more spaced the characters are, the less ambiguous the seed is likely to be. In [3], we describe the distribution of “resolution lengths,” which is one of the ways to measure this “independence” property. .

In models where there are only substitution differences, the differences are independent and uniformly distributed, and each seed is examined independently of other seeds, gapped and ungapped seeds have the same probability of error. However, this is not the case with indels, because any indel in the range of the seed, even an indel that occurs in a gap in the seed, corrupts the seed.

# 3 Algorithms

## 3.1 Algorithm 1: Coupled-seeds

We denote by $S_0^{(m_1,m_2,g)}$ a seed with the pattern:

$$S_0^{(m_1,m_2,g)} = (1, 2, ...m_1,\ m_1 + g + 1,\ m_1 + g + 2, ...,\ m_1 + g + m_2).$$

We refer to the first $m_1$ characters of the seed as "subseed 1" and to the last $m_2$ characters as "subseed 2". We refer to the $g$-long gap between the subseeds as "seed gap".

The seed has $m_1 + m_2$ characters from a range of $m_1 + m_2 + g$ characters in a string. Due to the long-range "independence" property in DNA, such seeds can be expected to be more unique than seeds of $m_1 + m_2$ consecutive characters.

These basic "coupled-seeds" are similar to any other gapped-seeds, with their various extensions. To use them, we create an index of all the occurrences of each of the possible $4^{m_1+m_2}$ seeds with the pattern $S_0^{(m_1,m_2,g)}$ in the reference. Then, we take several seeds in the query read $Y$ and look for their occurrences in the index, following standard hash-table algorithms.

### 3.1.1 Indels in coupled-seeds

We observe that the original coupled-seeds $S_0^{(m_1,m_2,g)}$ are not vulnerable to substitution differences that occur in the seed gap. However, they are vulnerable to indels. An indel of $d$ characters in the seed gap causes the character that should have appeared at position $m_1 + g + j$ to appear at $m_1 + g + j + d$, where $d$ can be a positive or negative integer. A combination of insertions and deletions can sum to 0, in which case the indels are "invisible" to the coupled-seed, like mismatches.

We denote by $S_d^{(m_1,m_2,g)}$ a seed with the pattern:

$$S_d^{(m_1,m_2,g)} = (1, 2, ...m_1,\ m_1 + g + d + 1,\ m_1 + g + d + 2,\ ...,\ m_1 + g + d + m_2)$$

where $d \geq -g$. We refer to these seeds as the "virtual variations of $S_0^{(m_1,m_2,g)}$."

To use the virtual variations, we take the original index, created for $S_0^{(m_1,m_2,g)}$, but create seeds from the query reads with several virtual variations. After the lists of hits for the different variations are created, we can follow standard hash-table algorithms.

For example, consider the reference $ACGTACT$ and the coupled-seed $S_0^{(2,2,2)} = (1, 2, 5, 6)$. The first seed in the perfect match $Y_1 = ACGTACT$ is $s_0^{(2,2,2)}(Y_1, 0) = (A, C, A, C)$, and it matches the reference seed. The read $Y_2 = ACaTACT$ has a mismatch, but it is "invisible" to the first seed. The read $Y_3 = ACTACT$ has an indel, and the first seed generated is $s_0^{(2,2,2)}(Y_3, 0) = (A, C, C, T)$, which does not match the reference. However, the virtual seed with $d = -1$ is $s_{-1}^{(2,2,2)}(Y_3, 0) = (A, C, A, C)$, exactly like the original seed in the reference.

An equivalent way of implementing "virtual seeds" is to introduce indels to parts of the read.

An alternative approach to "virtual seeds" is to store all the virtual variations in a single large index. In other words, the index for the reference contains multiple variations of the seed, all stored in the same list. This approach requires more memory, but allows faster search. Clearly, hybrids of the two approaches are also possible.

The "multiple index entries" approach to coupled-seeds is related to an idea that has been used in paired de Bruijn graphs for assembly[11].

## 3.2 Algorithm 2 : Pre-alignment filtration

We receive a read Y of length M. We would like to obtain a small selection of candidate locations in $W$, where the read is likely to be aligned with a small edit distance.

We select a small number of well separated, large segments of the read Y. For example, in a long read of length 3000, we choose the first 1000 characters to be the segment $Y_1$ and the last 1000 characters to be the segment $Y_2$. In paired-end reads we choose the the two reads as the two segments.

For each segment, we follow a seed based approach: we choose a number of seeds in the segment and build a list of hits for each segment. The list $C_1$ corresponds to the candidates for $Y_1$ and the list $C_2$ corresponds to candidates for the segment $Y_2$.

Next, we create the list $\tilde{C}_1$, which contains entries of $C_1$ for which there are entries in $C_2$ within an appropriate distance. Similarly, we create the filtered list $\tilde{C}_2$.

We use a combination of two methods for this filtration:

- Hash tables of regions where there are hits in $C_1$ and hash tables of regions where there are hits in $C_2$. The hashed tables of hits for $C_1$ is used to filter isolated hits from the list $C_2$, and vice versa.

- Sorted lists of $C_1$ and $C_2$. A simple iteration over the lists of hits allows us to filter out isolated candidates.

The filtered lists contain hits from relatively few regions of the genome. These regions are the regions where the reads are likely to be mapped to, and the hits are candidate anchors for alignment in these regions.

This approach can easily be extended from a constraint on two separate segments to constraints on more segments. It can also be extended to constraints "along a diagonal," requiring a number of hits which are appropriately separated in the read and in the genome (without explicit segments).

We note that paired-end reads have been used for filtration in RazerS in a q-gram based algorithm. RazerS first uses filters to produce “candidate matches” from multiple “seed hits.” It then discards of “matches” that do not appear as pairs. Here, we propose a direct use of the seeds without constructing “dot plots.”

In implementations, q-gram based algorithms typically break large references into separate segments and treat each segment separately. The “seeds” used in q-gram algorithms are typically shorter. An implementation of the algorithm presented here, can be used to index and analyze large references without such iterations over all the segments. The algorithm presented here is adequate when longer seeds are feasible, and when more memory is available for processing.

### 3.3 Combining the algorithms

Clearly, the two algorithms are independent. Algorithm 1 describes an extension of “seeds” and algorithm 2 describes a method of using arbitrary seeds for filtration. It is also clear that the two algorithms can be easily combined, by implementing the pre-filter of algorithm 2 with the particular coupled-seeds of algorithm 1.

## 4 Implementation

We implemented two versions of a pre-alignment algorithm in C. The first version was designed for paired-end reads, and the second version was designed for long reads. The implementations were tested using simulated reads from human reference genomes. The references were not masked to remove repetitive regions (beyond the original masking by “N”s). Any masking of repetitions would significantly accelerate the algorithms.

The implementations were tested in single thread mode on a cluster node with (2) E5620 CPUs and 48GB RAM.

These implementations are simple “proofs of concept,” and they are not optimized. In particular, the RAM requirement is flexible and can be easily reduced.

### 4.1 Paired-end implementation

The paired-end version of the pre-alignment algorithm was set to find all the possible candidates for an edit distance of up to 4 in each of the two 100bp reads in paired-end reads (8 in the two reads together).

In this case, we use 5 non-overlapping seeds in each of the reads. A list of hits, denoted by $C_1$, is created for the first read in the pair. A similar list $C_2$ is created for the second read. We create a hashed list of regions where $C_1$ has hits, and create a subset of the entries in $C_2$ which originate from adjacent regions. The same procedure is used to create a subset of $C_1$. The subsets are then sorted and refined with a more precise condition on the gap between the hits.

The seeds used in this implementation are "hybrid" random permutation seeds [12]. Standard seeds may be more efficient then the hybrid seeds in this particular algorithm.

To test the algorithm, we simulated $10^6$ pairs of reads, each 102bp long. 4 indels were introduce to each read (8 indels per pair). For a full, unmasked, human reference genome, and pairs with 250bp separation, the algorithm produced an average of 137 candidate per pair in 15.6 minutes. For a 1000bp separation, an average of 30 candidates was produced in 15.1 minutes. For a smaller reference, human chromosome 1, $10^6$ pairs of reads, separated by 250bp were processed in 1.75 minutes, producing 13.4 possible candidates on average. We examined the list produced for each of the reads and determined whether one of the candidates corresponded to the region from which the read had been generated (where "regions" are about 1000bp long). When one of the candidates corresponds to the correct region, we say that the alignment was correct. The alignment was correct in all the cases.

## 4.2 Long reads implementation

The long-reads version of the pre-alignment algorithm was designed as a randomized algorithm, which allows very high indel rates in relatively long reads ($> 1000$bp), like "PacBio" reads.

In this case, we distribute a number of seeds along the read. We create a list of all the hits, and a hashed list of the regions in which the hits appear. We remove isolated hits from the list by removing regions that were not "hit" more than twice. Finally, we sort the remaining hits and require each valid hit to have another hit separated by about $0.3 - 1$ times the length of the read. The seeds used here were a "hybrid" random permutation[12] version of coupled-seeds.

To test the algorithm, we simulated $10^6$ reads of 3500bp each (3.5Gbp). Random insertion and deletions were added with at a rate of 5% each (together, 10% indel rate), and substitution were added at a 1% rate. For each of the reads, the algorithm produced a list of candidates. We examined the list produced for each of the reads and determined whether one of the candidates corresponded to the region from which the read had been generated (where "regions" are about as long as a read). When one of the candidates corresponds to the correct region, we say that the alignment was correct. We measured the run time and the number of correct alignments. We also measured the average number of candidates in order to verify that the list is indeed "small."

The results of several runs for the full unmasked human reference are shown in figures 1 and 2. The results of several runs for the chromosome 1 reference are shown in figures 3 and 4. Figures 2 and 4 show the number of correct alignments as a function of the run-time. Figures 1 and 3 show the number of correct alignment as a function of the average number of candidates which the algorithm produced for each read. Each run represents a different choice of parameters.

Preliminary analysis of intermediate results in the different stages of filtration, and of the results for different parameters, suggests that even more basic filters are be sufficient for some applications. Higher degrees of accuracy and smaller lists of candidates can be obtained by optimizing the filters proposed here.

Interestingly, given a set of parameters, both implementations of the algorithm process reads with more errors slightly faster than they process reads with fewer errors. This is a result of the special distribution of strings in the DNA. Since a small fraction of the DNA is repeated many times, seeds from error-free reads can often be mapped to many different locations. These few seeds that are repeated a huge number of times become very significant in the overall processing time of algorithms. The algorithms described in this report reduce the ambiguity that such seeds produce, but they do not eliminate it completely. When errors are introduces into the seeds, they corrupt some of the repetitive seeds and replace them with rare seeds. Since rare seeds are processed faster, the algorithms are faster when there are more errors in the reads.

Figure 1: Complete human genome: number of candidates and % of reads aligned correctly

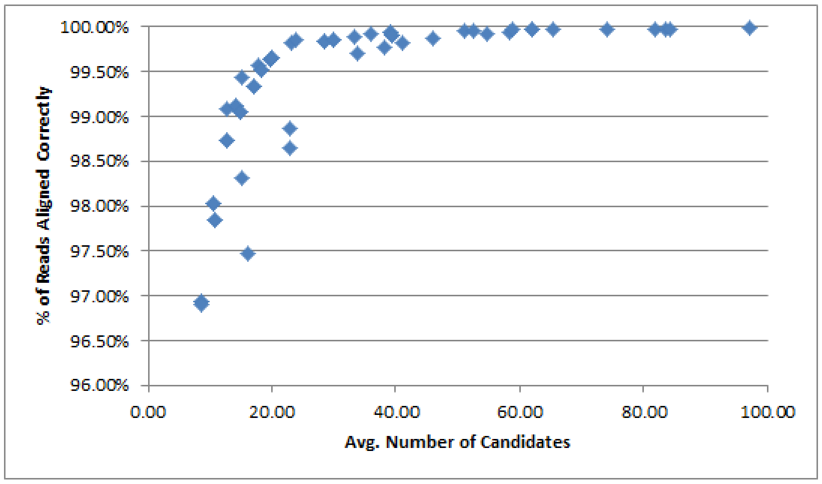


Figure 2: Complete human genome : search time and % of reads aligned correctly

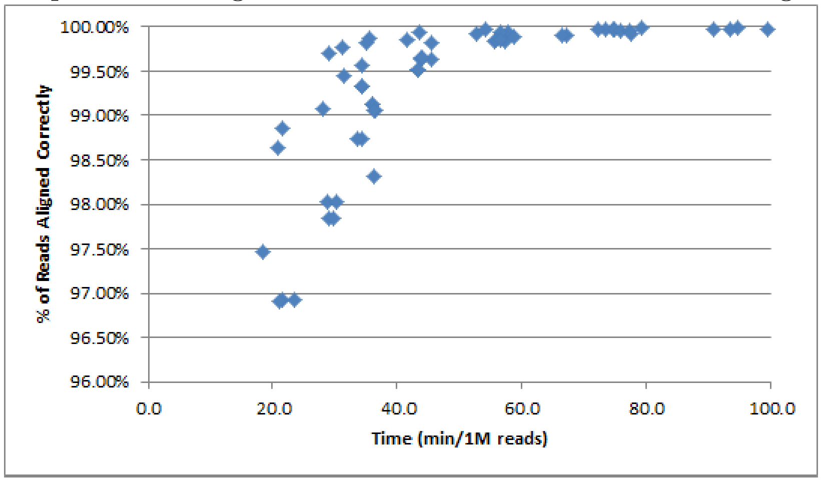


Figure 3: Chromosome 1 : number of candidates and % of reads aligned correctly

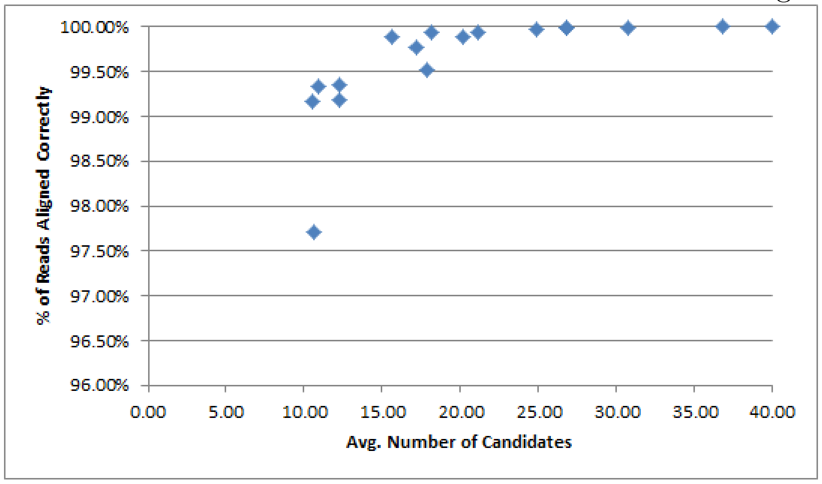
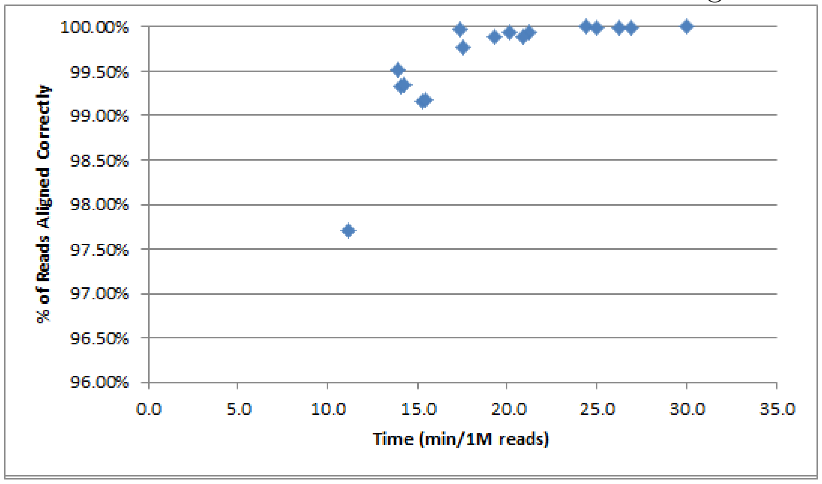


Figure 4: Chromosome 1 : search time and % of reads aligned correctly



## 5 Conclusions

Two algorithms have been constructed to use a long range "independence" property of DNA. The algorithms have been combined to generate short lists of candidate alignments for long reads with high indel probabilities. The two underlying algorithms are independent and be used separately.

The coupled-seeds algorithm provides an extension to seeds. It permits gapped seeds that span a long range and which are therefore not very ambiguous, but also allows indels in the span of the seed. Coupled-seeds can be used to extend existing seed-based algorithms, but they may require some adaptations in the implementations.

The pre-alignment filtration algorithm provides means of filtering long lists of seed hits by cross-referencing hits from different sections of reads, or pairs of reads. Pre-alignment algorithms can be used with any type of seed, and with paired-end reads as well as single reads of various lengths.

The pre-alignment approach is proposed as a preliminary step in alignment. In long reads, and moderate-size genomes, it is likely to produce a single candidate for each read or a very small number of candidates for each read. In large scale problems, some of the reads are likely to have multiple candidate alignments, but the number of candidates is significantly smaller than the number of candidates examined in seed-based algorithms without similar filters. After the list of candidates is generated, a refinement step is required, where any other alignment algorithm can be used to choose the appropriate candidates from the list.

The algorithms were demosntrated in basic implementations. These implementations are proofs of concept and they can be greatly improved and customized for particular applications.

The implementations were tested with full, unmasked genomes (with the exception of the "N" masking in the original reference). Alignment algorithms sometimes use some degree of masking of repetitive regions. Any such masking would accelarate the algorithm very significantly.

The idea of the pre-alignment algorithm can be extended to overlap search for clustering and assemly.

## 6 Acknowledgments

We would like to thank Knut Reinert, Miklós Csürös, Roy Ronen and Farhad Hormozdiari for their help. We would also like to thank the Yale University Biomedical High Performance Computing Center (NIH grants RR19895 and RR029676-01).

## References

[1] R L. A random-permutations-based approach to fast read alignment. *BMC Bioinformatics*, 14(Suppl 5):S8, 2013.

[2] B. Ma, J. Tromp, and M. Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, March 2002.

[3] R L. A note about the resolution-length characteristics of DNA. 2013.

[4] S F Altschul, W Gish, W Miller, E W Myers, and D J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, October 1990. PMID: 2231712.

[5] Can Alkan, Jeffrey M Kidd, Tomas Marques-Bonet, Gozde Aksay, Francesca Antonacci, Fereydoun Hormozdiari, Jacob O Kitzman, Carl Baker, Maika Malig, Onur Mutlu, S Cenk Sahinalp, Richard A Gibbs, and Evan E Eichler. Personalized copy number and segmental duplication maps using next-generation sequencing. *Nature Genetics*, 41(10):1061–1067, August 2009.

[6] D. Weese, A.-K. Emde, T. Rausch, A. Doring, and K. Reinert. RazerS–fast read mapping with sensitivity control. *Genome Research*, 19(9):1646–1654, July 2009.

[7] Petteri Jokinen and Esko Ukkonen. Two algorithms for approxmate string matching in static texts. In Gerhard Goos, Juris Hartmanis, and Andrzej Tarlecki, editors, *Mathematical Foundations of Computer Science 1991*, volume 520, pages 240–248. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

[8] Stefan Burkhardt, Andreas Crauser, Paolo Ferragina, Hans-Peter Lenhof, Eric Rivals, and Martin Vingron. q-gram based database searching using a suffix array (QUASAR). pages 77–83. ACM Press, 1999.

[9] Kim R. Rasmussen, Jens Stoye, and Eugene W. Myers. Efficient q-gram filters for finding all -matches over a given length. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Satoru Miyano, Jill Mesirov, Simon Kasif, Sorin Istrail, Pavel A. Pevzner, and Michael Waterman, editors, *Research in Computational Molecular Biology*, volume 3500, pages 189–203. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[10] Birte Kehr, David Weese, and Knut Reinert. STELLAR: fast and exact local alignments. *BMC Bioinformatics*, 12(Suppl 9):S15, 2011.

[11] Paul Medvedev, Son Pham, Mark Chaisson, Glenn Tesler, and Pavel Pevzner. Paired de bruijn graphs: A novel approach for incorporating mate pair information into genome assemblers. *Journal of Computational Biology*, 18(11):1625–1634, November 2011.

[12] R L. Adaptive hybrid permutation seeds (in preparation).