Designing Seeds for Similarity Search in Genomic DNA

Jeremy Buhler
jbuhler@cse.wustl.edu
Department of Computer Science and Engineering
Washington University
St. Louis, MO 63130
Uri Keich
keich@cs.cornell.edu
Department of Computer Science
Cornell University
Ithaca, NY 14853
Yanni Sun
yanni@cse.wustl.edu
Department of Computer Science and Engineering
Washington University
St. Louis, MO 63130

Abstract

Large-scale comparison of genomic DNA is of fundamental importance in annotating functional elements of genomes. To perform large comparisons efficiently, BLAST [3, 2, 4] and other widely used tools use seeded alignment, which compares only sequences that can be shown to share a common pattern or "seed" of matching bases. The literature suggests that the choice of seed substantially affects the sensitivity of seeded alignment, but designing and evaluating seeds is computationally challenging.

This work addresses the problem of designing a seed to optimize performance of seeded alignment. We give a fast, simple algorithm based on finite automata for evaluating the sensitivity of a seed in a Markov model of ungapped alignments, along with extensions to mixtures and inhomogeneous Markov models. We give intuition and theoretical results on which seeds are good choices. Finally, we describe **Mandala**, a software tool for seed design, and show that it can be used to improve the sensitivity of alignment in practice.

1 Introduction

Genomes and genomic sequence databases provide a fundamental reference tool for molecular biologists. These databases are used primarily to search for DNA sequences similar to (i.e. differing by few mutations from) a query sequence, or for pairs of sequences similar to each other. Applications of similarity search include detecting repetitive elements [35] and noncoding parts of genes, augmenting the power of gene-structure prediction [23], comparing whole genomes [13], and identifying sequences of unknown origin or function. Public genomic DNA sequence databases such as GenBank are growing exponentially [28], driving demand for fast comparison algorithms and heuristics that nonetheless are as sensitive as possible to biologically meaningful sequence conservation.

Seeded alignment is the dominant paradigm for accelerating large-scale genomic sequence comparison. BLAST [3, 2, 4] and other widely used tools [34, 21] apply alignment algorithms like

Smith-Waterman [36] only to pairs of sequences that exhibit prior evidence of similarity in the form of a shared *seed*, typically a common short substring or *word* of matching bases[1]. All matching words between two sequences can be found quickly, so seeded alignment efficiently directs computational resources toward pairs of sequence regions most likely to exhibit high similarity. The words in a sequence database can also be statically indexed [8, 21] to accelerate subsequent searches for word matches.

While words are the most popular type of seed for seeded alignment, discontiguous patterns of matching bases have seen considerable use in the sequence comparison literature. A discontiguous pattern spanning s bases, unlike a word of length s, requires matching pairs of bases at only a subset of the positions $\{0, 1, \ldots s-1\}$. Califano and Rigoutsos, in their FLASH comparison tool [8], found that randomly chosen discontiguous patterns in practice yielded the highest sensitivity to pairs of similar sequences when used to index a database. Buhler [6] formally established the sensitivity of random patterns in developing the randomized LSH-ALL-PAIRS comparison algorithm. Discontiguous patterns have also been used to accelerate seeded alignment algorithms including that of Pevzner and Waterman [31] and, more recently, the BLASTZ algorithm [34, 33]; Ma, Tromp, and Li's PatternHunter [26, 25]; and work by Brejova et al. [5].

PatternHunter introduced an important formal innovation to seeded alignment: the *resource-constrained paradigm* of seed design. This paradigm fixes the computational cost of seeded alignment *a priori* by fixing the number of different seeds to be used and the approximate false positive rate for each seed. It then asks how to choose seeds that maximize the probability of detecting ungapped alignments described by a probabilistic model.

The resource-constrained paradigm of seed design is well-suited to BLAST-like tools, in which the cost of using more than one or a few seeds to search a database is unacceptably high, as well as to static indexing schemes in which the number of indices, and hence of seeds, can be larger but is constrained by storage and disk access costs. However, actually designing seeds in the resource-constrained paradigm, even for simple probabilistic alignment models, is computationally challenging. Moreover, most existing work on resource-constrained seed design (with the notable exception of [5]) does not consider alignment models more informative than an i.i.d. random sequence of matches and mismatches.

This work describes tools for resource-constrained seed design. We address the following problem:

> *Given a collection of ungapped genomic sequence alignments of fixed length ℓ, whose distribution of matching base pairs is described by a kth-order Markov model $\mathcal{M}$, and resource limits w and n, find n seeds $\pi_1 \ldots \pi_n$, each inspecting w bases, such that the* ***sensitivity****, or probability that at least one seed detects a random alignment from $\mathcal{M}$, is maximized.*

This problem formalizes the data-driven design of seeds for alignment. The cost of filtering is controlled in two ways. First, we fix the *weight* of, or number of positions inspected by, each seed, which largely controls its false positive rate, i.e. the chance of seeing a seed match in the absence of an ℓ-mer alignment. Second, we fix the total number of seeds permitted. The Markov model $\mathcal{M}$ describes a model of "interesting" ungapped alignments that can be adjusted to reflect the empirical statistics of alignments between DNA sequences of a particular type (e.g. protein-coding).

In the following sections, we present theoretical and practical results on solving resource-constrained seed design and show that the design problem for models of order greater than zero

[1] BLASTN, unlike BLASTP, does *not* compute a neighborhood of each word in the query because typical word lengths are much longer for DNA than for protein.

is of practical interest in improving algorithms for biosequence similarity search. Section 2 gives an exact algorithm to compute detection probabilities for sets of seeds in Markov models. This algorithm, which uses dynamic programming on a finite automaton, generalizes and accelerates the original algorithm devised for the design of PatternHunter's seed. Section 3 investigates the relative detection probabilities of different seeds for ℓ-mer alignments and elucidates the structure of seed space. We first give an intuitive account of which properties are desirable in a seed, then prove the existence of seeds that are asymptotically optimal with increasing length ℓ. Section 4 describes Mandala, a tool to design near-optimal seeds for alignment models derived empirically from a collection of biosequences. Section 5 presents controlled trials using human-mouse genomic sequence comparisons to illustrate the practical utility of discontiguous seeds and of our design methods. Finally, Section 6 concludes and indicates directions for future work.

2 An Exact Algorithm for Seed Detection Probabilities

In this section, we formally define both the design space of potential seeds for alignment and a measure of goodness by which to evaluate elements of this space. Although an appropriate measure of goodness – sensitivity to alignments drawn from a probabilistic model – is conceptually straightforward, evaluating this measure efficiently is computationally challenging. Section 2.2 therefore describes an efficient evaluation algorithm that uses dynamic programming on finite automata derived from one or more seeds.

2.1 Problem Definition

Let C be a collection of genomic sequences. We seek all "interesting" ungapped alignments of some fixed length ℓ between pairs of substrings of C. In a BLAST-like algorithm, each such alignment serves as a starting point for gapped extension. An ungapped alignment between a pair of ℓ-mers consists of ℓ pairs of bases, each of which may be a match or a mismatch. Alternatively, it may be viewed as a string of ℓ bits, with a 1 wherever two bases match and a 0 where they fail to match. The bit-string representation of alignments discards some potentially useful information, such as their frequencies of transitions versus transversions, but it is sufficient to model the behavior of seeds, such as those used by BLAST and PatternHunter, that do not distinguish among different matched or mismatched base pairs.

An alignment is modeled by a kth-order Markov process $\mathcal{M}$ that gives the probability that the next bit seen will be 1 (i.e. a matching pair of bases) given the values of the previous k bits. The zeroth-order marginal probabilities of $\mathcal{M}$ describe the alignment's overall degree of conservation, while its higher-order marginals can reflect specific patterns of conservation. For example, alignments in coding sequence often exhibit a pattern of two matches followed by a mismatch, corresponding to conservation of the underlying protein with silent mutations at third base positions of codons.

Our goal is to devise a *seed* π, which is an ordered list of w positions $\{x_1 \ldots x_w\}$. We call w the *weight* of π, also denoted $|\pi|$, while its *span* is the distance $x_w - x_1 + 1$. We say that π *detects* an alignment S if, for at least one offset j, $S[j + x_i] = 1$ for $1 \leq i \leq w$. That is, every position of S inspected by π at offset j must contain matching bases. Without loss of generality, we require that $x_1 = 0$, since a seed with $x_1 > 0$ is equivalent (up to boundary conditions) to a seed of the same weight but shorter span. Seeded alignment algorithms enumerate all pairs of locations in the input C that exhibit matching bases in the pattern prescribed by π, so they are guaranteed to detect every alignment that π detects. If we instead devise a set Π of $n > 1$ patterns, then Π detects

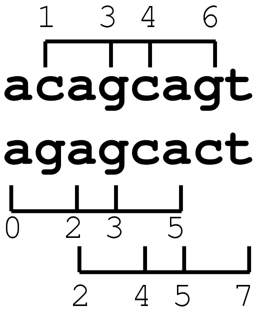

Figure 1: applying a seed, in this case $\{0, 2, 3, 5\}$, at multiple offsets can cause it to inspect overlapping sets of sequence positions.

alignment S if at least one of its component patterns detects S.

The computational cost of using seeds is controlled by their weight weight w, which largely determines their false positive rate in the absence of a meaningful alignment, and by the number n of seeds used. Following [26], we assume that the investigator sets these parameters *a priori* to match available computing resources. The problem, then, is to find a set Π of n seeds of weight w that maximizes sensitivity to ℓ-mer alignments S from model $\mathcal{M}$. That is, we want the set Π that maximizes the *detection probability*

$$\Pr_{S \sim \mathcal{M}} [\Pi \text{ detects } S] .$$

In practice, the alignment length ℓ is less than 100 bases; it represents the typical distance between indels in the alignments of interest. BLAST-like algorithms typically use a single seed of weight $w = 10$–15.

2.2 Computing Detection Probabilities

Figure 1 illustrates the key difficulty in computing detection probabilities accurately enough to differentiate among seeds of a given weight. A seed is applied at all possible offsets into an alignment, and we wish to compute the probability that it matches at *at least one* such offset. The probability of at least one match varies with the seed structure because matches at sufficiently close offsets do not occur independently. For example, the alignment in the figure has two matches, at offsets 0 and 2, which share two of four positions in common. The precise pattern of overlaps induced by a seed determines the dependencies in an i.i.d. alignment model; for more general Markov models, matches at non-overlapping offsets are also non-independent.

We cannot simply ignore overlap in analyzing seeds. Seed matches in biosequence alignments are not (and should not be!) rare events, so Poisson approximation does not accurately estimate detection probabilities. We could instead approximate a seed's detection probability using partial information about its overlap structure, e.g. by using the first couple of terms in the inclusion-exclusion form of the detection probability. However, for seeds of practically interesting weight and span, it is possible to efficiently compute the *exact* detection probability analytically.

Before proceeding, we pause to ask whether an analytical approach to match probabilities is the right choice in practice. We could instead generate pseudorandom alignments from the model $\mathcal{M}$ and estimate how often such alignments contain the desired seed. This Monte Carlo approach is easy to implement and generalizes to *any* generative alignment model. In practice, however, it has the significant drawback that the number of random alignments required to accurately estimate match probabilities is quite high. To obtain estimates accurate to one part in a thousand requires in excess of 10^6 trials. The computational cost of Monte Carlo estimation is such that the exact algorithm described below proved at least two orders of magnitude faster for the seeds designed in this work.

Our algorithm to compute detection probabilities encodes the overlap structure of a seed or set of seeds into a deterministic finite automaton (DFA). Considering alignments as bit strings, a seed π describes a certain pattern of 1's in those strings. We compute the probability of seeing π by first constructing a DFA $\mathcal{A}_\pi$ that accepts all strings containing this pattern, then computing the probability that $\mathcal{A}_\pi$ accepts a string chosen at random from the model $\mathcal{M}$.

DFAs have previously been used to summarize the dependence between matches at overlapping offsets by Nicodéme et al. [29] and by Tompa [39], who used them to compute occurrence probabilities for degenerate words in sequences. In contrast, methods for computing seed detection probabilities, including the one originally devised for PatternHunter and later modifications by Keich et al. [20], Brejova et al. [5], and Choi and Zhang [11], use dynamic programming in a way that does not explicitly represent the overlap structure. We find that considering the DFA explicitly both makes the evaluation algorithm easier to reason about and exposes opportunities for optimization of our methods.

2.2.1 DFA Construction

Let π be a seed with weight w and span s, and let Q_π be the set of all 2^{s-w} s-bit strings that match π. Following [29], we could construct a small nondeterministic finite automaton accepting all strings containing a match to Q_π, then use the subset construction [19] to compile it into the desired DFA $\mathcal{A}_\pi$. This approach is reasonable when it would be too difficult to construct $\mathcal{A}_\pi$ directly, though the exponential worst-case behavior of the subset construction means that it might be slow in practice. However, $\mathcal{A}_\pi$ has an efficient direct construction.

First, construct a trie T_π from the strings of Q_π. T_π may be viewed as a DFA that accepts precisely the language Q_π; its root is the start state, while each of its leaves is an accepting state. We next convert T_π to a DFA $\mathcal{A}_\pi^0$ that accepts any input containing a string from Q_π as a suffix. $\mathcal{A}_\pi^0$ can be built efficiently from T_π using the Aho-Corasick algorithm [1], which adds *failure links* to T_π that indicate, for any state corresponding to an input string α, the state corresponding to the longest proper suffix of α. If we find that we cannot follow a trie edge out of state s on a given input, we instead follow the path of failure links out of s until we *can* continue with a trie edge, or until we reach the start state. Once the failure link for state s is known, we follow it if needed to determine the proper transition out of s on a 0 bit. (Note that for non-accepting states, we never fail on seeing a 1 bit.) If we add failure links to T_π in breadth-first order from the start state, the full path of such links out of state s is well-defined when s is first processed.

The desired DFA $\mathcal{A}_\pi$ must accept every input containing a string of Q_π as a *substring*, not just as a suffix. To form $\mathcal{A}_\pi$ from $\mathcal{A}_\pi^0$, we make each accepting state of the latter an absorbing state, so that the DFA accepts forever the first time it sees a seed match. All accepting states are now equivalent and so may be collapsed into a single state q_a.

We have observed empirically that, for the seeds designed in this work, the DFA $\mathcal{A}_\pi$ is 3–30 times larger than the smallest DFA accepting its language.

2.2.2 Probability Computation

By construction, the automaton $\mathcal{A}_\pi$ accepts an alignment S, represented as a bit string, iff π detects S. The following dynamic programming algorithm computes the probability that $\mathcal{A}_\pi$ accepts a random alignment of length ℓ from a kth-order Markov model $\mathcal{M}$. We give a description that applies without change to any $k \geq 0$, though a slightly more space-efficient implementation is possible if the cases of $k = 0$ and $k > 0$ are treated separately.

Let δ be a bit string of length k. For a state q, let $\Phi_b(q)$ be the set of all states that transition to q on bit b. Define $P(q, t, \delta \cdot b)$ to be the probability that the automaton $\mathcal{A}_\pi$ reaches state q after reading t bits of input S, *and* that the last $k + 1$ bits read form the *history* string $\delta \cdot b$. We may derive

$$P(q, t, \delta \cdot b) = \Pr(S[t] = b \mid S[t - k \ldots t - 1] = \delta) \times \sum_{q' \in \Phi_b(q)} \sum_{b_0 \in \{0,1\}} P(q', t - 1, b_0 \cdot \delta). \qquad (1)$$

In other words, to reach state q with history $\delta \cdot b$ at time t, the automaton must first reach state q' with history $b_0 \cdot \delta$ at time $t - 1$, then execute a transition from q' to q on bit b. The transition probability $\Pr(S[t] = b \mid S[t - k \ldots t - 1] = \delta)$ is given by the model $\mathcal{M}$. For $1 \leq t \leq k$, the history $b_0 \cdot \delta$ is shortened to length t, and the transition probabilities are given by $\mathcal{M}$'s lower-order marginals.

To initialize the recurrence, we set $P(q_0, 0, 0) = 1$ for the start state q_0 and set all other probabilities for $t = 0$ to 0. This initialization ensures that $P(q, 1, b) = \Pr(S[1] = b)$ if a transition from q_0 to q on b exists, or 0 otherwise. Once we have completed ℓ steps of the recurrence, the final probability that $\mathcal{A}_\pi$ has reached its accepting state q_a is the sum over all $k + 1$-bit strings $\delta \cdot b$ of $P(q_a, \ell, \delta \cdot b)$.

We note that the Markov model constructed by Choi and Zhang [11, Section 2.3] is essentially the same as our DFA. However, they use their model only to obtain analytical formulas for the match probabilities of certain well-behaved seed families, while we use our DFA as the core of our dynamic programming algorithm for any seed.

2.2.3 Efficiency

The size of T_π, and hence of $\mathcal{A}_\pi$, is surely at most $s2^{s-w}$, the total length of all strings in Q_π. When the strings of Q_π are generated from a common seed, the size of the trie can be more precisely bounded as follows.

Lemma 1. *The trie T_π has size at most $(w + 1)2^{s-w}$.*

Proof. The trie built for a seed π of span s has a single root followed by s levels of nodes. Number the root as level 0 and the remaining levels as $1 \ldots s$. The portion of the trie up to level i encodes all bit strings of length i that are prefixes of some string accepted by π. If π inspects position $i - 1$ (counting from 0), then every string matching π has a 1 in position $i - 1$, and so trie level i has the same number of nodes as level $i - 1$. Otherwise, strings matching π can have a 0 or a 1 in position $i - 1$, and so level i has *twice* as many nodes as level $i - 1$.

Consider the trie built for the following seed of weight w: $\pi^* = \{0, s-w+1, s-w+2, \ldots, s-1\}$. This seed has span s and inspects its first position and its last $w-1$ positions. The seed π^* therefore matches all s-bit strings of the form $1(0|1)^{s-w}1^{w-1}$. The number of nodes at each level of the trie T_{π^*} therefore follows the progression

$$1, 1, 2, 4, \ldots, 2^{s-w}, \ldots, 2^{s-w}$$

where the last value is duplicated $w-1$ times after its initial appearance, once for each of the $w-1$

terminal required 1 bits. The total size of T_{π^*} is therefore

$$\begin{aligned}
|T_{\pi^*}| &= 1 + \sum_{i=0}^{s-w} 2^i + (w-1)2^{s-w} \\
&= 1 + 2 \cdot 2^{s-w} - 1 + (w-1)2^{s-w} \\
&= (w+1)2^{s-w}.
\end{aligned}$$

We now claim that every other seed π of weight w and span s results in a trie of size less than $|T_{\pi^*}|$. There are still 2^{s-w} strings matching π, so T_π still has 2^{s-w} leaves. Moreover, we require by definition that π has $x_1 = 0$, so the progression of level sizes in T_π still begins with "1, 1". However, if $\pi \neq \pi^*$, then some number $j \geq 1$ of π's required 1 bits occur prior to last $w-1$ bits. Hence, the progression of level sizes for T_π contains j fewer instances of 2^{s-w} and a corresponding number of earlier duplicated values, each strictly less than 2^{s-w}. We conclude that T_π has strictly fewer nodes than T_{π^*}. □

The Aho-Corasick construction runs in time linear in the trie size, as does the collapsing of the final states into q_a, so $\mathcal{A}_\pi$ can be built from π in worst-case time $\Theta(w2^{s-w})$. Although some states of $\mathcal{A}_\pi$ may have many parents, the total number of parents over the entire DFA is simply its total number of transitions, which is at most twice the number of states. Hence, each step of dynamic programming takes time $\Theta(w2^{s-w}2^k)$ for a kth-order model. The full computation therefore runs in time $\Theta(w2^{s-w+k}\ell)$. This time is faster by a factor of s/w than the algorithms of [11] and [20], which work only for model order $k = 0$. A C++ implementation of our algorithm, applied with $\ell = 64$, a seed π with $w = 11$ and span $s = 18$ (the same ℓ, w, and s used by PatternHunter), and order $k = 5$, runs in a few tens of milliseconds on a 2.5 GHz Intel Pentium IV workstation.

Using the construction of the previous section, the time required to build the DFA $\mathcal{A}_\pi$ is negligible compared to the cost of computing the dynamic programming recurrence. The latter cost scales with the number of states in $\mathcal{A}_\pi$, so it is often advantageous to minimize $\mathcal{A}_\pi$ prior to computing the match probability. Because the DFAs for patterns of practical weight and span have thousands or tens of thousands of states, minimization must be done in time subquadratic in the number of states to yield an overall speedup. We use Hopcroft's $n \log n$ minimization algorithm [18].

We note that, using a slightly more complex construction, our algorithm's cost can be reduced to $\Theta((2^k + w2^{s-w})\ell)$ (see Appendix A for details). However, we did not implement this last speedup because it relies on the underlying trie structure of the DFA $\mathcal{A}_\pi$ and so will not work if the DFA is minimized prior to dynamic programming.

2.2.4 Extensions

The above algorithm assumes a single seed π and a single kth-order Markov alignment model $\mathcal{M}$. However, it is straightforward to extend the computation to support multiple seeds or somewhat more complex alignment models.

- To compute the probability that at least one seed in a set Π matches a random alignment, begin the DFA construction with the set $Q_\Pi = \bigcup_{\pi \in \Pi} Q_\pi$, and proceed as before.
- Although we use fixed-length alignments in our analysis, one could instead extend $\mathcal{M}$ to include a distribution over alignment length. The dynamic programming algorithm of Section 2.2.2 could be extended to compute the detection probability for $\mathcal{M}$ as a weighted sum of probabilities for its component lengths.

- Consider a *mixture model*, in which alignments are drawn randomly from one of several Markov models $\mathcal{M}_z$ with associated mixture priors p_z. Then the match probability for a seed π is given by
$$\Pr_{S\sim\mathcal{M}}\left[\pi \text{ detects } S\right] = \sum_z p_z \Pr_{S\sim\mathcal{M}_z}\left[\pi \text{ detects } S\right].$$
- Consider an *inhomogeneous Markov model*, in which the probability of seeing a given bit at position t of an alignment depends not only on the last few bits seen but on t itself. A good example is an alignment of coding DNA, where the probability of seeing a matching base pair depends strongly on the sequence position modulo three. To compute detection probabilities for a given DFA in such a model, we simply draw from the appropriate position-specific distribution for t when computing $\Pr(S[t] = b \mid \ldots)$ in Equation (1).

To design seeds appropriate to coding DNA, we model alignments using a mixture of three Markov models, representing pairs of aligned ℓ-mers starting in frame 0, 1, or 2 respectively, with equal (or empirically derived) priors. Each model is inhomogeneous with period three, reflecting the three-periodic pattern of coding sequence conservation.

3 The Structure of Seed Space

When is one seed more sensitive than another? The answer seems to depend in a complicated way on both the parameters of $\mathcal{M}$ and the alignment length ℓ. For example, while PatternHunter's seed outperforms a contiguous word π_c of the same weight in a zeroth-order model with $\ell = 64$ and 70% identity, it can be shown that for short enough ℓ or sufficiently low identity[2], π_c becomes optimal. Such parameter-dependent irregularities complicate comparisons among seeds. In this section, we first consider what, if any, formal statement we can make about the superiority of some classes of seeds over others, then provide some intuitive explanation of this phenomenon.

For an alignment S of length ℓ, define $E_\ell(\pi)$ to be the event that π detects S at *some* offset, and define $E^c_\ell(\pi)$ to be the complementary event. (We drop the π, writing E_ℓ, when π is clear from context.) We will explore the zeroth-order alignment model postulated in [26].

The following claim demonstrates that at least *some* general comparisons are possible among classes of seeds. Call a seed *uniformly spaced* if its positions form an arithmetic progression with difference > 1, e.g. $\{0, 2, 4, 6, \ldots\}$. Recall that π_c is a contiguous word.

Claim 1. *If π is a uniformly spaced seed, then, for any ℓ and any zeroth-order model $\mathcal{M}$,*
$$\Pr[E_\ell(\pi)] < \Pr[E_\ell(\pi_c)].$$

Proof. Denote by $B_j(\pi)$ the event that the uniformly spaced seed π matches an alignment S at offset j, i.e. $B_j(\pi) = \{S[j + x_i] = 1\ ,\ \forall x_i \in \pi\}$. To simplify notations, we only consider a 2-periodic seed $\pi = \{0, 2, 4, \ldots, 2(w-1)\}$, though the same proof would hold for any uniformly spaced seed. Write $\ell = 2m + \delta$, where $\delta \in \{0, 1\}$. Let $D_\pi = \cup_{j=1}^{m+\delta-(w-1)} B_{2j-1}(\pi)$ (π matches S at odd indices) and $F_\pi = \cup_{j=1}^{m-(w-1)} B_{2j}(\pi)$ (π matches at even indices). Similarly, define $F_{\pi_c} = \cup_{j=1}^{m-w} B_j(\pi_c)$, define $D_{\pi_c} = \cup_{j=m+1}^{\ell-(w-1)} B_j(\pi_c)$, and define $G_{\pi_c} = \cup_{j=m-w+1}^{m} B_j(\pi_c)$.

[2] The latter result follows by inspecting the inclusion-exclusion form of the detection probability.

We have that $\Pr[D_\pi] = \Pr[D_{\pi_c}]$ and $\Pr[F_\pi] = \Pr[F_{\pi_c}]$. Moreover, D_π is independent of F_π, and the same holds for π_c, so

$$\begin{aligned}\Pr[E_\ell(\pi)] &= \Pr[D_\pi] + \Pr[F_\pi] - \Pr[D_\pi]\Pr[F_\pi] \\ &= \Pr[D_{\pi_c} \cup F_{\pi_c}] \\ &< \Pr[D_{\pi_c} \cup F_{\pi_c}] + \Pr[G_{\pi_c} \setminus (D_{\pi_c} \cup F_{\pi_c})] \\ &= \Pr[E_\ell(\pi_c)].\end{aligned}$$

□

3.1 Asymptotic Structure of Seed Space

While some comparisons among seeds, like Claim 1, hold for any fixed alignment length ℓ, others, like the apparent optimality of the contiguous seed π_c, hold only for small ℓ. To avoid irregularities that occur for small ℓ, we turn to sensitivity measures that hold *asymptotically*, that is, as ℓ grows large. Asymptotic results can elucidate properties of seed space that are not apparent for small ℓ, and we may hope that they apply at least approximately to the range of ℓ assumed in practice.

The following result shows that the asymptotic performance of a seed is well-defined.

Claim 2. *For any seed π there exist $\beta > 0$ and $\lambda > 0$ (both of which depend on π) such that $\Pr[E_\ell^c]/\lambda^\ell \longrightarrow \beta$ as $\ell \to \infty$.*

Proof. For this and following results, we need the following *matrix representation* of the DFA $\mathcal{A}_\pi$, after [29]. We assume here that the model $\mathcal{M}$ is zeroth-order, relegating the proof of the kth-order case to Appendix A. Let $N + 1$ be the number of states of the DFA, and let A_π be an $N \times N$ matrix indexed by all states of $\mathcal{A}_\pi$ except its accepting state, such that $A_\pi(s,t)$ is the probability of transitioning from state s to state t of the DFA upon reading a base pair generated from $\mathcal{M}$. Note that A_π is sparse, entrywise non-negative, and sub-stochastic.

Let $\boldsymbol{e}_1 = (1, 0, 0, \ldots, 0) \in \mathbb{R}^N$. Then $\boldsymbol{e}_1 A$ yields the distribution of the automaton states one step (or one bit) after starting from the entry state q_0. More generally, $\boldsymbol{e}_1 A^\ell$ yields the distribution on the states after ℓ steps[3]. This non-negative distribution has total mass less than 1; indeed, the difference is exactly $\Pr[E_\ell]$.

Assume for a moment that A is primitive in the sense that there exists a j such that A^j is an entrywise strictly positive matrix. Since A^j is a sub-stochastic positive matrix, by the Perron-Frobenius theorem it has a multiplicity-1, positive eigenvalue $\lambda^j < 1$ such that any other eigenvalue μ of A^j satisfies $|\mu| < \lambda^j$ [24]. Moreover, the left eigenvector $\boldsymbol{u}$ corresponding to λ^j is positive, and without loss of generality $\boldsymbol{u}$'s entries sum to 1. As $\boldsymbol{u}$ is, up to a constant, the unique left eigenvector of A^j with eigenvalue λ^j and since A and A^j commute, $\boldsymbol{u}$ is also a left eigenvector of A: $\boldsymbol{u}A = \lambda\boldsymbol{u}$. Similarly, there exists a corresponding positive right eigenvector $\boldsymbol{w}$ for which $A\boldsymbol{w} = \lambda\boldsymbol{w}$. Let $\boldsymbol{e}_1 = \beta\boldsymbol{u} + \boldsymbol{v}$, where $\boldsymbol{v}$ is a linear combination of the other (possibly generalized) left eigenvectors of A^j. Then

$$0 < \boldsymbol{e}_1\boldsymbol{w} = \beta\boldsymbol{u}\boldsymbol{w} + \boldsymbol{v}\boldsymbol{w} = \beta\boldsymbol{u}\boldsymbol{w},$$

and, since $\boldsymbol{u}\boldsymbol{w} > 0$, $\beta > 0$. Thus, assuming A is primitive we have

$$\boldsymbol{e}_1 A^\ell = \beta\lambda^\ell\boldsymbol{u} + \boldsymbol{v}A^\ell = \beta\lambda^\ell\boldsymbol{u} + \mathrm{o}(\lambda^\ell),$$

[3] This formulation introduces another way to compute $\Pr[E_\ell]$ in time $\mathrm{O}(w2^{s-w}\ell)$ by noting that $\boldsymbol{e}_1 A^n = (\boldsymbol{e}_1 A^{n-1})A$ [39].

which immediately implies the claim.

Finally, notice that for a seed π of span s, any non-terminal state of $\mathcal{A}_\pi$ is accessible from the entry state by at most s steps and vice versa. Because we can stay at the entry state for an arbitrary number of steps, A^{2s} is entrywise positive, whence A is primitive. □

Remark 1. *A version of Claim 2 can be traced back to [29] with many missing details which are provided here for completeness.*

Definition 1. *A seed π is asymptotically worse than a seed π', denoted $\pi \prec \pi'$, if*

$$\lim_{\ell} \Pr[E^c_\ell(\pi)] / \Pr[E^c_\ell(\pi')] > 1.$$

Similarly, π and π' are asymptotically equivalent, denoted $\pi \simeq \pi'$, if

$$\lim_{\ell} \Pr[E^c_\ell(\pi)] / \Pr[E^c_\ell(\pi')] = 1,$$

and $\pi \preceq \pi'$ if $\pi \prec \pi'$ or $\pi \simeq \pi'$.

Remark 2. *Using Claim 2, it is easy to verify that this relation defines a linear order on the set of all seeds. A seed π's asymptotic performance is entirely determined by $\lambda(\pi)$ and $\beta(\pi)$.*

We now use our ability to define the asymptotic performance of seeds to argue that the widely used contiguous seed is not very promising for large ℓ. From now on, we assume S is generated by a zeroth-order model.

Claim 3. *Let π_c be the contiguous seed of weight w and π any seed with $|\pi| = w$. Then*

$$\lambda(\pi_c) \geq \lambda(\pi). \tag{2}$$

Proof. In [20, Claim 1] the authors prove that

$$\Pr[E_\ell(\pi_c)] \leq \Pr[E_{\ell+s-w}(\pi)].$$

It follows from Claim 2 that

$$\lim_{\ell\to\infty} \frac{\beta(\pi_c)\lambda(\pi_c)^\ell}{\beta(\pi)\lambda(\pi)^{\ell+s-w}} \geq 1,$$

which proves the claim. □

We conjecture that equality holds in (2) if and only if π is a uniformly spaced seed. The "if" part follows immediately from Claim 1 and the preceding claim[4]. If this conjecture is true, than any non-uniformly spaced seed is asymptotically better than the contiguous seed. While we cannot yet prove this conjecture, we *can* show that π_c cannot be asymptotically optimal:

Claim 4. *Let $\pi = \{0, 1, \ldots, w-2, w\}$ where $|\pi| = |\pi_c| = w > 2$. Then $\lambda(\pi_c) > \lambda(\pi)$, and in particular $\pi_c \prec \pi$.*

[4] It is not hard to show, in the spirit of Claim 1, that for any uniformly spaced seed π, $\pi \prec \pi_c$; that is, although $\lambda(\pi) = \lambda(\pi_c)$, $\beta(\pi) > \beta(\pi_c)$.

Proof. Let $\boldsymbol{x} = (x_1, \dots, x_N)$ be the unique unit-mass positive eigenvector for which $\boldsymbol{x}A(\pi) = \lambda(\pi)\boldsymbol{x}$, where $A(\pi)$ is the nonnegative automaton matrix associated with π and $N = w+2$ is its dimension. Without loss of generality, the automaton of π consists of states F_i, $i = 1 \dots w$, which correspond to a prefix of $i-1$ 1 bits, and states F_{w+1} and F_{w+2}, which correspond to a prefix of $w-1$ consecutive 1 bits followed by a 0 and a 1 respectively. For example, with $w = 3$ and $q = 1-p$, $A(\pi_{c_3}) = \left(\begin{smallmatrix} q & p & 0 \\ q & 0 & p \\ q & 0 & 0 \end{smallmatrix}\right)$, while $A(\pi) = \begin{pmatrix} q & p & 0 & 0 & 0 \\ q & 0 & p & 0 & 0 \\ 0 & 0 & 0 & q & p \\ q & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & q & 0 \end{pmatrix}$.

From $\boldsymbol{x}A(\pi) = \lambda(\pi)\boldsymbol{x}$ we obtain the following system of equations:

$$q(x_w + x_{w+2}) = \lambda(\pi)x_{w+1} \qquad px_w = \lambda(\pi)x_{w+2},$$

from which we deduce that

$$\lambda(\pi)^2 x_{w+1} = q(\lambda(\pi) + p)x_w. \tag{3}$$

Consider the polynomial $f(\lambda) = \lambda^2 - q(\lambda + p)$. For $p \in (0, 1)$ (and $q = 1 - p$) it has a unique positive root, and it happens that $f(\lambda)$ is the characteristic polynomial of the automaton matrix for π_{c_2}: $A(\pi_{c_2}) = \left(\begin{smallmatrix} q & p \\ q & 0 \end{smallmatrix}\right)$. Since $w > 2$, $\pi_{c_2} \subset \pi$, and so the unique positive root of $f(\lambda)$, $\lambda(\pi_{c_2})$, satisfies $\lambda(\pi_{c_2}) \le \lambda(\pi) \le \lambda(\pi_c)$. Suppose for a moment that $\lambda(\pi_{c_2}) < \lambda(\pi_c)$. In this case either $\lambda(\pi) < \lambda(\pi_c)$, which is exactly our claim, or $\lambda(\pi_{c_2}) < \lambda(\pi)$. The latter implies that $f(\lambda(\pi)) > 0$, and it follows from (3) that $x_w > x_{w+1}$.

Note that $\boldsymbol{x}A(\pi) = \lambda(\pi)\boldsymbol{x}$ implies that for $i = 1, 2, \dots, w-1$, $px_i = \lambda(\pi)x_{i+1}$ and $q(x_{w+1} + \sum_{i=1}^{w-1} x_i) = \lambda(\pi)x_1$. Thus, with $\hat{\boldsymbol{x}} = (x_1, \dots, x_w)$ ($\boldsymbol{x}$ without its last two digits),

$$\hat{\boldsymbol{x}}A(\pi_c) = \left(q\sum_{i=1}^{w} x_i, px_1, px_2, \dots, px_{w-1}\right) = (\lambda(\pi)x_1 + q(x_w - x_{w+1}), \lambda(\pi)x_2, \dots, \lambda(\pi)x_w)\,.$$

Since we showed that $q(x_w - x_{w+1}) > 0$ it follows exactly as in the proof of the Perron-Frobenius theorem (e.g. [24]) that $\lambda(\pi_c) > \lambda(\pi)$ as stated.

Finally, to show that $\lambda(\pi_{c_2}) < \lambda(\pi_c)$, we note that since $w \ge 3$ it suffices to check that $\lambda(\pi_{c_2}) < \lambda(\pi_{c_3})$, which can readily be verified using symbolic algebra software such as Maple. □

Remark 3. *While one might be suspicious of the practicality of asymptotic results, in this case one can prove that if $w \ge 4$ then for $\ell = w + 3$, $p > \frac{1}{2}$, and the seed π given in Claim 4,* $\Pr[E_\ell(\pi)] > \Pr[E_\ell(\pi_c)]$. *The same inequality holds if $w \ge 5$, $\ell = w + 4$, and $p > \frac{1}{3}$.*

3.2 On good and bad seeds

Consider the following problem studied by Conway [17] among others. The input is a random string of letters, each generated uniformly (each of the letters is equally likely) and independently of the others. We are looking for two particular words in the text: α=`AAAAA` and β=`ABCDE`. At any specific location in the string, the probability of seeing α is the same as that of seeing β: $1/26^5$. Similarly, renewal theory guarantees that the average distance between overlapping occurrences of α is the same as that between occurrences of β: 26^5 (e.g. [32]). Faced with this apparent symmetry, one would be tempted to guess that we are equally likely to see the first occurrence of α before that of β as to see first the occurrence of β before that of α. However, this intuition is false: we are more likely to see β before α.

This confusing fact becomes more plausible when one considers clustered occurrences. Given that α is observed starting at position i, we can get another occurrence at position $i+1$ at a huge discount: $1/26$ instead of the usual $1/26^5$. Occurrences of β, on the other hand, cannot overlap, and therefore it pays retail for every occurrence. Thus, relative to the occurrences of β, the occurrences of α tend to appear in clustered blocks. Since the *average* distance between occurrences of both words is the same it follows that the average distance between the blocks of α has to be larger than the average distance between the blocks of β (which contain a single occurrence of β). This exactly translates to longer waiting time[5] for the first occurrence of α than for that of β.

It is essentially this phenomenon that is behind the advantage a (properly) spaced seed has over the contiguous seed. The occurrences of the contiguous seed exhibit a tendency to cluster when compared with those of the spaced seed, while the average distance between occurrences of both seeds in a randomly generated string is the same: $1/p^w$. For the contiguous seed this follows from renewal theory, as in the example above, while for the spaced seed, one needs the equivalent result from Markov renewal theory [9].

So which seeds are good? It is not hard to prove that, *were* the occurrences of a certain seed independent, that seed would be optimal, at least in our asymptotic framework. Of course, the occurrences of a seed can never be independent due to their overlap, but we should try to minimize the overlap to create seeds whose occurrences are as independent as possible. In general that task would conflict with our desire to keep the span of the seed under control when considering a region of a finite length, and some compromise between the two has to be struck. That conflict is, however, removed when considering asymptotically optimal seeds. In that case, we empirically find that the best seeds are the ones which have at most one bit in common for any shift (and are the minimal spanwise with that property). These seeds are exactly optimal Golomb rulers [14] and represent the best approximation for independent occurrences. A Golomb ruler for $w = 6$, for example, is $\{0, 1, 8, 11, 13, 17\}$.

4 Mandala: Software for Seed Design

The results of Section 3 provide insight into the structure of seed space but do not immediately lead to a practical algorithm for picking the best seed for a given alignment model $\mathcal{M}$. This problem was recently shown to be NP-hard even for a zeroth-order alignment model [25]. Xu et al. [42] give a (somewhat compute-intensive) approximation algorithm for finding seeds that are within a bounded factor of the optimal sensitivity, but the bound obtained depends in a complicated way on the seed weight and span and can be fairly weak. Heuristic methods remain the approach of choice for fast, practical design of seeds. To address the need for fast seed design in practice, we have constructed Mandala, a software tool utilizing heuristic methods, along with the algorithm of Section 2, to design seeds for seeded alignment.

At a high level, Mandala is organized as follows. The user provides a collection of unaligned sequences. We first search these sequences for pairwise local alignments associated with annotated features of some specified type, e.g. coding exon, UTR, or *Alu* repeat. We then use the alignments found to parameterize an alignment model $\mathcal{M}$, which captures the essential properties of the alignments for use in seed design. Finally, we search seed space, subject to certain constraints such as a fixed weight w, to find the best seed(s) for detecting alignments drawn from $\mathcal{M}$. This section considers practical issues in parameterizing the model $\mathcal{M}$ and describes our local search algorithm for seed selection.

[5] Arriving at a random time to a train station, which train line are we more likely to see departing first: one that has 5 trains departing one per minute for the first 5 minutes after the hour, or one that has 5 trains departing at 12-minute intervals?

4.1 Model Parameterization

The models considered in this work describe ungapped alignments of fixed length ℓ. Given a set of such alignments, we may parameterize a kth-order Markov model $\mathcal{M}$ by translating each alignment into a bit string (1 for match, 0 for mismatch) and counting the frequencies of all $k+1$-bit substrings in the training set. In practice, the challenge lies not in computing $\mathcal{M}$'s parameters but in obtaining an appropriate set of training alignments.

Training alignments can be obtained by extracting ungapped segments from alignments involving the sequence features of interest. The typical length of these segments dictates the alignment length ℓ for the model. We have consistently used $\ell = 64$ in this work because ungapped segments of this length or longer commonly occur in statistically significant alignments of human and mouse genomic DNA[6]. The required alignments could be obtained from the input sequences using any of numerous extant similarity search tools, but some caution is required in selecting a tool to avoid unduly biasing the training set. If, for example, we were to use a seeded alignment tool such as BLASTN that finds only alignments containing a particular seed π, our training set would assign a higher match probability to π than is warranted by the underlying sequences.

An unbiased approach to finding training alignments would use a search algorithm that does not rely on seeds at all. This approach is technically possible given the existence of robust dynamic programming search tools, in particular the Sim algorithm [43], that return multiple nonoverlapping alignments from a single pair of sequences. However, such tools take a long time to search the megabases of DNA needed to produce a reasonably sized training set. We therefore took a more efficient compromise approach to training set generation that nonetheless limits seed bias. Buhler's LSH-ALL-PAIRS algorithm [6] finds ungapped ℓ-mer alignments in a collection of sequences using repeated seeded alignment with a large number of seeds chosen independently and uniformly at random. Seeds are allowed to have span up to ℓ. We sampled alignments from our human and mouse sequences using LSH-ALL-PAIRS with tens of different randomly chosen seeds, so that no one or a few seeds would be strongly favored in the training data. This approach can sample tens or hundreds of thousands of ℓ-mer alignments from 10^7 bases of genomic sequence in a few tens of minutes on a single workstation, or from 10^8–10^9 bases of sequence in an hour or two on a modest computing cluster.

A second issue in choosing training alignments is how well-conserved they should be. Highly conserved alignments (85% identity or more) are poor choices for training a model $\mathcal{M}$ because the performance of different seeds becomes nearly indistinguishable; in essence, any seed will do for an "easy" search problem. Better seed discrimination is obtained by training on the least conserved alignments that are still considered interesting. We typically train our models on alignments of 70–75% identity. When sampling alignments with a range of different identities, we use rejection sampling to counteract the bias of seeded alignment toward more conserved alignments.

A final consideration in training is whether one can reliably sample only the alignments of interest. For example, biologically meaningful alignments of coding DNA sequences should be "in frame;" that is, an aligned pair of bases should always be from the same codon position in the two aligned sequences. However, we have found that simply sampling alignments of 70-75% identity between annotated human and mouse coding sequences produces numerous alignments that are *not* in frame, hence not biologically meaningful and not useful for training. The solution is either to eliminate the meaningless alignments from the training set, or to use an alternative similarity search algorithm that does not produce them. In this work, we take the second approach, following [5]. We use NCBI TBLASTX [3] to find ungapped local alignments of amino acid sequences in the six-frame

[6] Our choice of ℓ is also historical – it is the length used in designing PatternHunter's seed [26].

translations of the input sequences, then draw our training set from the aligned DNA sequences underlying these proteomic alignments. Although TBLASTX does not take the same degree of care as LSH-ALL-PAIRS to produce an unbiased sample of alignments, it does avoid BLASTN's reliance on exact word matching (at either the DNA or protein level).

4.2 Finding Optimal Seeds

Given the alignment model $\mathcal{M}$, Mandala seeks a seed (or set of seeds) with the highest possible detection probability subject to a fixed seed weight w, which roughly controls a seed's false positive rate and hence the computational cost of using it. Other than brute-force enumeration and evaluation, no procedure is known to find the optimum seed of fixed weight w and span up to some s for a model $\mathcal{M}$. Enumeration is fast enough to choose single seeds in simple alignment models but rapidly becomes impractical as the number of simultaneous seeds, the model complexity, or the maximum span increases. Mandala therefore sacrifices global optimality for much faster design using the following *local* search method.

Let $\pi = \{x_1 \ldots x_w\}$ be the current seed, with all $x_i \leq s$. As usual, we fix $x_1 = 0$ to avoid generating shifted versions of the same seed. The local neighborhood of π is the set of all seeds π' that differ from π in exactly one of $x_2 \ldots x_w$, with the differing position chosen from among the unused set $\{1 \ldots s-1\} - \pi$. Using this neighborhood definition and the probability calculation of Section 2 as an evaluation function, we perform hill climbing with random restart in seed space to find a near-optimal seed.

To design a set of simultaneous seeds Π, we can extend the neighborhood definition to encompass all sets Π' in which one seed $\pi'_i \in \Pi'$ differs from the corresponding $\pi_i \in \Pi$ in a single position. However, this method exhibits slow convergence in practice, so Mandala implements a greedy approach that progressively picks seeds. Details of the method, which empirically seems to work as well as simultaneous design, are given in [37]. An independent implementation of greedy multiseed design was also recently described in [25].

The speedup of Mandala over exhaustive seed enumeration is dramatic, even for problems in which enumeration is feasible. Using the same design constraints as Ma, Tromp, and Li ($\ell = 64$, $w = 11$, zeroth-order model with 70% identity [26]), we sought the best seed of span $s \leq 22$. Mandala with ten random restarts finished in under 20 seconds on a 2.5 GHz Intel Pentium IV workstation, versus over an hour for exhaustive enumeration. In ten trials of ten restarts each, Mandala found the globally optimal seed three times; the seven suboptimal seeds found differed from the optimum in their detection probabilities by less than 1%.

The evaluation algorithm of Section 2 is fast for small spans, but its cost grows exponentially as the span increases for fixed weight. We therefore heuristically limit the search space of seeds to those with span $s \ll \ell$, usually permitting spans up to $2w$. In practice, this limitation has not proven problematic. However, if Mandala must consider seeds with larger spans, it can fall back on Monte Carlo estimation of detection probabilities, as suggested in Section 2.1. Monte Carlo estimation uses a large number of pseudorandom bits and is therefore highly sensitive to bias in the values produced by the underlying pseudorandom number generator. We have obtained good results with the Mersenne Twistor generator [27], though more common long-period generators such as the GNU C Library's `random()` function [15] proved inadequate. The optimizer accelerates Monte Carlo evaluation for hill-climbing search using Wald's Sequential Probability Ratio Test [40] to rapidly terminate evaluation of moves with a high probability of being downhill.

5 Experimental Results

Mandala's design criteria produce seeds that work well in theory, but we naturally sought further evidence that these seeds are also useful in practice. We investigated the following questions. First, do discontiguous seeds, and Mandala's seeds in particular, *by themselves* benefit seeded alignment? Second, does empirical evidence support our conjecture in Section 3 that contiguous seeds are among the *least* sensitive choices? Third, how well do our Markov alignment models $\mathcal{M}$ predict seeds' actual performance?

The performance of an alignment algorithm depends not only on seed choice but also on a variety of other factors, such as choice of score matrix and aggressiveness in extending seed matches into gapped and ungapped alignments. Programs such as PatternHunter that improve several aspects of alignment at once do not therefore provide clear proof of the utility of seed design. We therefore devised the following test setup to isolate the effect of seed choice on performance. Starting with assemblies of the human and mouse genomes obtained from the UCSC Genome Browser [22], we extracted 1262 pairs of regions that had been annotated as syntenic fragments, i.e. orthologous regions with no major internal rearrangements. These regions, spanning roughly 2.65 gigabases, were masked to remove repeats and low-complexity DNA, then divided into coding and noncoding parts on the basis of annotated coding exons predicted by the Twinscan gene structure prediction program [23]. Our experiments measured the number of nonoverlapping gapped alignments found when comparing orthologous pairs of regions.

Sequences were compared using BLAST-like seeded alignment. We modified the `lsh` program from the Projection Genomics Toolkit [7] to use a fixed externally specified seed or set of seeds. The program performed gapped extension using banded Smith-Waterman with the default scoring function and significance threshold used by PipMaker [34], whose parameters are optimized for human-mouse comparison. Other than the choice of seed, no property of the alignment tool was changed between experiments. Appendix B gives further details of our experimental setup.

5.1 Noncoding DNA Sequence

We used roughly 1.4 million ungapped alignments sampled from our syntenic regions, with $\ell = 64$ and 70–75% identity, to parameterize a fifth-order Markov alignment model $\mathcal{M}_5$. The choice of model order was somewhat arbitrary, though we had sufficient data to parameterize it; we therefore evaluated seeds generated not only from the full model but also from each of its lower-order approximations, which we denote $\mathcal{M}_0 \ldots \mathcal{M}_4$. The zeroth-order approximation $\mathcal{M}_0$ is identical to the model of [26] except with $p \approx 0.724$ instead of 0.7. In the interest of time, we tested seed performance using only a subset of 449 pairs of syntenic fragments, spanning roughly 500 megabases of unmasked sequence.

We concentrated on weight-11 seeds, since this weight is commonly used in NCBI BLASTN. Figure 2 illustrates the theoretical average detection probability for such seeds as a function of their span. The figure supports our conjecture that nearly *any* seed is more sensitive than the contiguous seed. Because the alignment length ℓ is finite, there are preferred spans for each model that optimize the tradeoff between maximizing the number of offsets available for detection and minimizing seed overlap between offsets. For $\mathcal{M}_0$ (solid line), the best spans are 17–19, consistent with the results of [26]; however, the full $\mathcal{M}_5$ (dashed line) prefers shorter spans of 13–15, suggesting that its alignments exhibit tighter grouping of their matching base pairs.

Table 1 gives the relative performance of several different seeds, both in theory and on our test set. For each seed, we give its detection probability in the fifth-order alignment model $\mathcal{M}_5$, the number of non-overlapping gapped alignments found using the seed, and the total CPU time

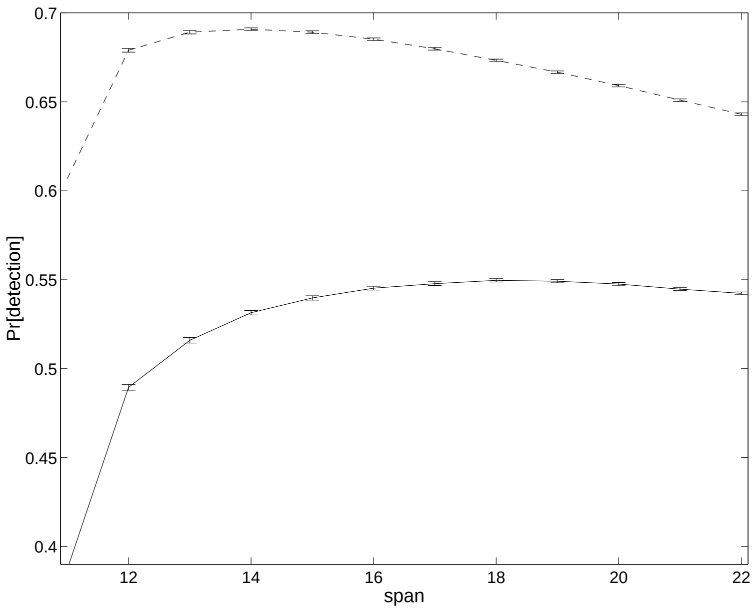

Figure 2: average detection probabilities given by theoretical models for random seeds of weight 11 and various spans. Solid line: zeroth-order noncoding model $\mathcal{M}_0$; dashed line: fifth-order model $\mathcal{M}_5$. Error bars are 95% confidence intervals for each average over 1000 trials.

required for the search. We first tested several known seeds: the contiguous 11-mer π_c used in NCBI BLASTN, the contiguous 10-mer π_{c10}, and the PatternHunter seed π_{ph}. We then compared the performance of the best seeds found by Mandala in ten restarts using each of the models $\mathcal{M}_0$ through $\mathcal{M}_5$, denoted in the table as π_{N_0} through π_{N_5}. All seeds were constrained to have weight 11 and span ≤ 22 (by which point sensitivity is apparently already declining according to Figure 2). We note that the best seeds found are predicted to be noticeably more sensitive than average seeds of the same span.

All discontiguous seeds found substantially more alignments in practice than did π_c; in particular, π_{ph} found 14.6% more alignments than π_c and 2.4% more than even π_{c10}. The extra alignments found were among the most difficult to detect, with scores up to a few times times the threshold score but far below the most "obvious" high-scoring alignments. Mandala's choice of seed for the zeroth-order model $\mathcal{M}_0$ behaved almost indistinguishably from π_{ph} in practice, demonstrating that its local search optimizer is a viable substitute for exhaustive enumeration of seeds.

The seeds chosen by Mandala for models $\mathcal{M}_1$ through $\mathcal{M}_5$ are instructive. Lower-order approximations to the full $\mathcal{M}_5$ yielded seeds that were close but theoretically inferior to π_{N_5} in performance. In practice, however, the performance of seeds π_{N_1} through π_{N_5} was nearly indistinguishable, suggesting that the model $\mathcal{M}_5$ actually contained little meaningful information for seed design above first order. The main difference between the zeroth-order $\mathcal{M}_0$ and higher-order models was that the latter showed a tendency for matches to occur in clumps rather than independently. This tendency is reflected in the shorter spans of the seeds induced from these models. A shorter span may be considered advantageous in practice because it increases the likelihood that a seed will detect an ungapped segment of an alignment with a large proportion of indels.

All discontiguous seeds incurred a slight speed penalty versus π_c, largely because their false positive rates (i.e. the number of seed matches that did not lead to a significant gapped alignment after extension) were up to 33% higher. However, the slowdowns observed were only about 11%, considerably less than the $> 30\%$ slowdown observed going from π_c to the shorter π_{c10}. Indeed,

Seed	Pattern	Pr[detection]	Alignments Found	Time
			(Thousands)	(mins)
π_c	$\{0,1,2,3,4,5,6,7,8,9,10\}$	0.607	220	382
π_{c10}	$\{0,1,2,3,4,5,6,7,8,9\}$	0.712	246	502
π_{ph}	$\{0,1,2,4,7,9,12,13,15,16,17\}$	0.689	252	417
π_{N_0}	$\{0,1,2,5,7,10,11,14,16,17,18\}$	0.680	252	417
π_{N_1}	$\{01,2,3,5,8,9,12,13,14,15\}$	0.699	252	423
π_{N_2}	$\{0,1,2,3,6,8,9,10,12,13,14\}$	0.707	253	424
π_{N_3}	$\{0,1,2,3,5,6,9,11,12,13,14\}$	0.704	252	422
π_{N_4}	$\{0,1,2,4,5,6,8,11,12,13,14\}$	0.707	253	425
π_{N_5}	$\{0,1,2,3,5,6,7,10,12,13,14\}$	0.709	253	424

Table 1: performance of seeds on noncoding DNA. Pr[detection] is detection probability in model $\mathcal{M}_5$. Gapped alignments found and running times are on 500 megabases of homologous noncoding regions from human and mouse.

the false positive rate for π_{c10} may be expected to be around 400% higher than for π_c, which (as observed in [26]) makes seed design a more attractive option to increase sensitivity than simply reducing a seed's weight.

Finally, we tested Mandala's ability to design simultaneous seeds by finding two seeds of weight 12, which should have sensitivity comparable to a seed of weight 11 while exhibiting roughly half the false positive rate. In accordance with our observations on single seed design, we used the first order model $\mathcal{M}_1$ to evaluate seeds. Mandala produced the following seed pair:

$$\begin{aligned}\pi_1 &= \{0,1,2,3,6,7,13,17,18,19,20,21\}\\ \pi_2 &= \{0,1,2,3,4,5,8,9,11,12,13,14\}.\end{aligned}$$

These two seeds together were 2.3% more sensitive than the best single seed (π_{N_5}) found but required roughly 16% more CPU time than it. The cost of finding seed matches dominates the practical cost of seeded alignment, so simultaneous seeds are most appropriate when indexing a sequence database offline or for hardware that can use multiple seeds in parallel. Even so, we note that the CPU time needed to use both 12-mer seeds was *still* less than that needed for π_{c10}.

5.2 Coding DNA Sequence

We tested seeds for coding DNA using annotated coding exons from all 1262 homologous region pairs. We extracted 357000 64-mer alignments with 70–75% identity from TBLASTX alignments of these regions, then used them to train the following model $\mathcal{M}_c$. To capture the codon structure in the sampled alignments, we first inferred three 5th-order Markov models $\mathcal{M}_{c1}$, $\mathcal{M}_{c2}$, and $\mathcal{M}_{c3}$ from aligned base pairs at first, second, and third codon positions respectively. These three models can be combined into one inhomogeneous model, in which the distributions used to generate successive positions cycle between the three codon positions. Three different inhomogeneous models are possible depending on whether the first position of an alignment is generated from $\mathcal{M}_{c1}$, $\mathcal{M}_{c2}$, or $\mathcal{M}_{c3}$. The final model $\mathcal{M}_c$ is a mixture of these three inhomogeneous models in the proportions encountered in the training set, which are roughly equal with a slight bias toward alignments starting at a first codon position.

A key design goal of Mandala is that it should exploit biologically meaningful structure present in its training alignments. For coding alignments, we expect that the best seed design should

Seed	Pattern	Pr[detection]	Alignments Found	Time
			(Thousands)	(mins)
π_c	$\{0,1,2,3,4,5,6,7,8,9,10\}$	0.442	77.8	77.6
π_{ph}	$\{0,1,2,4,7,9,12,13,15,16,17\}$	0.627	92.2	81.0
π_{110}	$\{0,1,3,4,6,7,9,10,12,13,15\}$	0.652	93.2	82.5
π_{C_5}	$\{0,1,2,8,9,11,12,14,15,17,18\}$	0.721	94.0	84.3

Table 2: performance of seeds on coding DNA. Pr[detection] is detection probability in model $\mathcal{M}_c$. Gapped alignments found and running times are on coding portions of 2.65 gigabases of homologous regions from human and mouse.

resemble a repeating "110" pattern that ignores every third position. Given the right offset into an alignment, such a seed would consider only the better-conserved first and second base positions of each codon. We investigated whether Mandala could automatically exploit this expected structure in $\mathcal{M}_c$.

Table 2 lists the seeds tested on our coding data set. Besides π_c and π_{ph}, we tried the intuitive coding seed π_{110} and the best weight-11 seed π_{C_5} found by Mandala in ten restarts on model $\mathcal{M}_c$ with span $s \leq 22$. The seed π_{C_5} is structured similarly but not identically to π_{110}.

Our tests show that, with the right alignment model, Mandala can not only validate but also improve on biological intuition. The seed π_{C_5} does reveal something like the expected "110" pattern. However, in this case biological intuition is somewhat in conflict with our intuition that highly periodic seeds are less sensitive (cf. Section 3.2). The seed π_{C_5} compromises between these two design constraints by breaking up the seed's periodicity without disrupting the underlying codon structure. The compromise seed is more sensitive in practice than either π_{ph}, which satisfies only the aperiodicity constraint, or π_{110}, which satisfies only the codon structure constraint.

6 Conclusions and Directions

Seed design materially affects the sensitivity of seeded alignment algorithms. Sensitivity improvements of even a few percent are worthwhile, given the heavy use of these techniques in practice (over 10^5 queries/day to the NCBI BLAST server). We have described algorithms and software tools to optimize the choice of seeds for particular alignment problems and have demonstrated that the right seed choice confers practical benefits. The Mandala software, with source code, may be obtained from *http://www.cse.wustl.edu/∼jbuhler/mandala/*.

Seed design is presently a fast-moving field, with extensive and continuing work [5, 37, 11, 25, 42] on obtaining the best seeds for pairwise alignment. Opportunities for substantial new contributions to bioinformatics from this technology now lie in extending the model-driven design of fast search heuristics to newer, more challenging problems, in particular to alignment of alignments and to RNA structural alignment.

As genomic sequences continue to proliferate, genomes of multiple species are increasingly being aligned into homologous blocks that span anywhere from a few hundred to a few million bases [30, 38]. These blocks seem destined to become a major component of sequence databases, and tools such as RPS-Blast [4] and PhyloCon [41] are already being developed to compare them to query sequences and to each other more efficiently. We plan to apply results from the design of seeds to these search problems. A key question is how to find efficient extensions of simple seed heuristics for these problems that also capture the information present in the distributions of residues in each column of a block.

Locally conserved structural motifs in RNA sequences may indicate a common mechanism for regulating gene expression. However, discovering such motifs, which may occur together with primary sequence conservation, requires simultaneous folding and local sequence alignment. A reasonable combinatorial formulation of this problem is known to be NP-hard for more than two sequences [12], so it is desirable to devise heuristics to scan a collection of long RNA sequences for short, shared structural motifs, much like PSI-BLAST [4] scans for potential shared primary sequence patterns. Again, initial scanning heuristics have been developed for this problem (see, e.g., [16]), but a more systematic exploration of heuristic space may be feasible using techniques developed for improving pairwise alignment.

Acknowledgments

The authors wish to thank Tom Joseph, for invaluable help with the infrastructure to run the experiments; Ming Li, Tomas Vinar, and Brona Brejova, for encouragement to work on finding asymptotically optimal seeds; Steven Altschul and Alejandro Schäffer, for helpful suggestions on Mandala's design; and anonymous referee for many helpful suggestions. Authors J.B. and Y.S. were supported by NSF CAREER award DBI-0237902.

References

[1] A. Aho and M. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18:333–40, 1975.

[2] S. F. Altschul and W. Gish. Local alignment statistics. *Methods: a Companion to Methods in Enzymology*, 266:460–80, 1996.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, et al. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–10, 1990.

[4] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–402, 1997.

[5] B. Brejova, D. G. Brown, and T. Vinar. Optimal spaced seeds for hidden markov models, with applications to homologous coding regions. In *Proceedings of the Fourteenth Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 42–54, 2003.

[6] J. Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17:419–28, 2001.

[7] J. Buhler. *Search Algorithms for Biosequences Using Random Projection.* PhD thesis, University of Washington, Seattle, WA, 2001.

[8] A. Califano and I. Rigoutsos. FLASH: a fast look-up algorithm for string homology. In *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology (ISMB '93)*, pages 56–64, 1993.

[9] E. Çinlar. Markov renewal theory. *Advances in Applied Probability*, 1:123–187, 1969.

[10] F. Chiaromonte, V. B. Yap, and W. Miller. Scoring pairwise genomic sequence alignments. In *Pacific Symposium on Biocomputing*, pages 115–26, 2002.

[11] K. P. Choi and L. Zhang. Sensitivity analysis and efficient method for identifying optimal spaced seeds. *Journal of Computing and System Sciences*, 68:22–40, 2004.

[12] E. Davydov and S. Batzoglou. A computational model for RNA multiple structural alignment. In *Proceedings of the Fifteenth Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 254–69, 2004.

[13] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, et al. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–76, 1999.

[14] A. Dewdney. Computer recreations. *Scientific American*, December:16–26, 1985.

[15] GNU Project. Gnu c library, 2003. ftp://ftp.gnu.org/gnu/libc.

[16] J. Gorodkin, S. L. Stricklin, and G. D. Stormo. Discovering common stem-loop motifs in unaligned RNA sequences. *Nucleic Acids Research*, 29:2135–44, 2001.

[17] L. Guibas and A. Odlyzko. String overlaps, pattern matching and nontransitive games. *Journal of Combinatorial Theory, series A*, 30:183–208, 1981.

[18] J. Hopcroft. *An $n \log n$ algorithm for minimizing states in a finite automaton*, pages 189–96. Academic Press, 1971.

[19] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MS, 1979.

[20] U. Keich, B. Ma, M. Li, and J. Tromp. On spaced seeds for similarity search. *Discrete Applied Mathematics*, 2003. In Press.

[21] W. J. Kent. BLAT: the BLAST-like alignment tool. *Genome Research*, 12:656–64, 2002.

[22] W. J. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and D. Haussler. The human genome browser at UCSC. *Genome Research*, 12:996–1006, 2002.

[23] I. Korf, P. Flicek, D. Duan, and M. R. Brent. Integrating genomic homology into gene structure prediction. *Bioinformatics*, 17 Suppl 1:S140–8, 2001.

[24] P. D. Lax. *Linear Algebra.* Pure and Applied Mathematics. John Wiley & Sons, Inc., New York, 1997.

[25] M. Li, B. Ma, D. Kinsman, and J. Tromp. PatternHunter II: highly sensitive and fast homology search. *Genome Informatics*, 14:164–75, 2003.

[26] B. Ma, J. Tromp, and M. Li. PatternHunter – faster and more sensitive homology search. *Bioinformatics*, 18:440–5, 2002.

[27] M. Matsumoto and T. Nishimura. Mersenne twistor: a 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.

[28] National Center for Biological Information. Growth of GenBank, 2004. http://www.ncbi.nih.gov/Genbank/genbankstats.html.

[29] P. Nicodéme, B. Salvy, and P. Flajolet. *Motif Statistics*, pages 194–211. Number 1643 in Lecture Notes in Computer Science. Springer, Berlin, 1999.

[30] P. Pevzner and G. Tesler. Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution. *Proceedings of the National Academy of Sciences USA*, 100:7672–7, 2003.

[31] P. Pevzner and M. S. Waterman. Multiple filtration and approximate pattern matching. *Algorithmica*, 13:135–54, 1995.

[32] S. Ross. *Stochastic processes*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., New York, 1996.

[33] S. Schwartz, W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller. Human-mouse alignments with BLASTZ. *Genome Research*, 13:103–7, 2003.

[34] S. Schwartz, Z. Zhang, K. A. Frazer, A. F. Smit, et al. PipMaker – a web server for aligning two genomic DNA sequences. *Genome Research*, 10:577–86, 2000.

[35] A. F. Smit and P. Green. Repeatmasker, 1999. http://ftp.genome.washington.edu/RM/RepeatMasker.html.

[36] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–97, Mar. 1981.

[37] Y. Sun and J. Buhler. Designing multiple simultaneous seeds for DNA similarity search. In *Proceedings of the Eighth Annual International Conference on Computational Molecular Biology (RECOMB04)*, pages 76–84, San Diego, CA, 2004.

[38] J. W. Thomas, J. W. Touchman, R. W. Blakesley, G. G. Bouffard, et al. Comparative analyses of multi-species sequences from targeted genomic regions. *Nature*, 424:788–93, 2003.

[39] M. Tompa. An exact method for finding short motifs in sequences, with applications to the ribosome binding site problem. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology (ISMB '99)*, pages 262–71, Heidelberg, Germany, 1999. AAAI Press.

[40] A. Wald. *Sequential Analysis*. Wiley, New York, 1947.

[41] T. Wang and G. D. Stormo. Combining phylogenetic data with co-regulated genes to identify regulatory motifs. *Bioinformatics*, 19:2369–80, 2003.

[42] J. Xu, D. G. Brown, M. Li, and B. Ma. Optimizing multiple spaced seeds for homology search. In *Proceedings of the Fifteenth Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 47–58, 2004.

[43] X. Xuang and W. Miller. A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics*, 12:337–57, 1991.

A Proofs Omitted from Text

A.1 Smaller Automaton Construction for Markov Case

The following construction builds a new DFA $\mathcal{B}_\pi$ equivalent to $\mathcal{A}_\pi$ such that each state can be reached by at most one possible k-bit history. First, note that every state of $\mathcal{A}_\pi$ of depth $\geq k$ in the trie T_π already has this property. For each state s of $\mathcal{A}_\pi$, let the *label* $\alpha(s)$ be the string labeling the path from the trie root to s.

Define new states s_j for each k-bit string j such that j is not $\alpha(s)$ for any state at depth k of T_π. Although these states are not in T_π, we define their labels $\alpha(s_j) = j$. On bit b, each new state s_j transitions to the state whose label $j' = j[2 \ldots k] \cdot b$ (either a new state $s_{j'}$ or an existing state of $\mathcal{A}_\pi$ whose label is j'). Moreover, if an existing state t of $\mathcal{A}_\pi$ with depth $\geq k$ transitions on bit b to a state at depth $< k$, that transition is changed to target the state $s_{j''}$, where the label j'' is the last $k-1$ bits of $\alpha(t)$ plus b. Now delete the (unused) old states of $\mathcal{A}_\pi$ with depth $< k$. Finally, augment the new DFA with a full tree of 2^{k-1} *initial* states, the root of which is the new start state, such that the leaf of the tree reached on input string α transitions on bit b to the state labeled $\alpha \cdot b$ (which may be an old or a new state).

We can divide the non-initial states s of the new $\mathcal{B}_\pi$ into equivalence classes $\Phi(t)$, where t indexes the states of $\mathcal{A}_\pi$. The relation is as follows: if s was a preexisting state t of $\mathcal{A}_\pi$, $s \in \Phi(t)$. For all new states s_j, $s_j \in \Phi(t)$ iff t is the state of $\mathcal{A}_\pi$ with depth $< k$ whose label forms the longest suffix of j. It can be shown that for any string S of length $\geq k$, if the original DFA $\mathcal{A}_\pi$ ends up in state s after reading S, then the new DFA $\mathcal{B}_\pi$ ends up in some state in $\Phi(s)$. Provided that $\mathcal{A}_\pi$ does not accept on any strings shorter than k bits, this proves equivalence of $\mathcal{A}_\pi$ and $\mathcal{B}_\pi$. Moreover, each state of the new DFA, except for the initial tree, is by construction associated with exactly one history of k bits.

Because each transition of $\mathcal{B}_\pi$ is associated with only one k-bit history, each transition has fixed probability in a kth-order Markov model. We can therefore define a (sparse) matrix A_π corresponding to $\mathcal{B}_\pi$ as described in the proof of Claim 2. The cost of computing sensitivity in the kth-order Markov model is therefore $O((2^k + w2^{s-w})\ell)$, the cost of ℓ matrix multiplications with the sparse A_π.

A.2 Proof of Claim 2 in the Markovian case

When the string S is governed by a kth-order Markov chain, we define the $N' \times N'$ matrix A' as the submatrix of the A from the construction of A.1 that corresponds to all non-initial states $\mathcal{B}_\pi$. A' is primitive since, as in the proof of the zeroth-order case, we can move between any two states in $2s$ steps. We can complete the proof as before upon replacing $\boldsymbol{e}_1$ with $\boldsymbol{\mu} \in \mathbb{R}^{N'}$, the distribution of the states after the first k bits. Note that the probability distribution, $\boldsymbol{\mu}$, is supported only on the states of $\mathcal{B}_\pi$ whose labels have length k and is readily obtainable from $\boldsymbol{e}_1 A^k$.

B Details of Experimental Setup

We obtained NCBI build 31 of the human genome and Release 2 of the mouse genome, along with their annotations, from the UCSC Genome Browser. Sequences were divided into annotated coding exons and the remaining noncoding DNA based on Twinscan's coding exon predictions. Soft-masked regions of the sequences, representing interspersed repeats and low-complexity DNA, were ignored for training and testing purposes.

To generate the coding training set, TBLASTX was run with the BLOSUM62 scoring function, modified so as to heavily penalize stop codons. Translated alignments with E-value at most 10^{-5} were mapped back to their underlying genomic sequences, and aligned segments with 70–75% identity were extracted for training.

Test comparisons were performed with a modified version of the `lsh` program. The program was modified to use hashing rather than sorting for detecting seed matches and to read a list of seeds from a file. Each seed match was subjected to ungapped extension using NCBI BLAST's linear-time dynamic programming algorithm, followed by gapped extension using banded Smith-Waterman. We scored alignments with the HOXD-70 score matrix [10] and affine gap penalties of -400 to open and -30 to extend. We kept only non-overlapping alignments that scored above 3000, the default cutoff used by PipMaker.