MPSCAN: Fast Localisation of Multiple Reads in Genomes

Eric Rivals, Leena Salmela, Petteri Kiiskinen, Petri Kalsi, and Jorma Tarhio

LIRMM, CNRS and Université de Montpellier 2, Montpellier, France
`rivals@lirmm.fr`
Helsinki University of Technology, P.O. Box 5400, FI-02015 TKK, Finland
`{lsalmela,iridian,tarhio}@cs.hut.fi`

Abstract. With Next Generation Sequencers, sequence based transcriptomic or epigenomic assays yield millions of short sequence reads that need to be mapped back on a reference genome. The upcoming versions of these sequencers promise even higher sequencing capacities; this may turn the *read mapping* task into a bottleneck for which alternative pattern matching approaches must be experimented. We present an algorithm and its implementation, called MPSCAN, which uses a sophisticated filtration scheme to match a set of patterns/reads exactly on a sequence. MPSCAN can search for millions of reads in a single pass through the genome without indexing its sequence. Moreover, we show that MPSCAN offers an optimal average time complexity, which is sublinear in the text length, meaning that it does not need to examine all sequence positions. Comparisons with BLAT-like tools and with six specialised read mapping programs (like BOWTIE or ZOOM) demonstrate that MPSCAN also is the fastest algorithm in practice for exact matching. Our accuracy and scalability comparisons reveal that some tools are inappropriate for read mapping. Moreover, we provide evidence suggesting that exact matching may be a valuable solution in some read mapping applications. As most read mapping programs somehow rely on exact matching procedures to perform approximate pattern mapping, the filtration scheme we experimented may reveal useful in the design of future algorithms. The absence of genome index gives MPSCAN its low memory requirement and flexibility that let it run on a desktop computer and avoids a time-consuming genome preprocessing.

1 Introduction

Next-generation sequencers (NGS), able to yield millions of sequences in a single run, are presently being applied in a variety of innovative ways to assess crucial biological questions: to interrogate the transcriptome with high sensitivity [1], to assay protein-DNA interactions at a genome wide scale [2], or to investigate the open chromatine structure of human cells [3,4]. Due to their wide applicability, cost effectiveness, and small demand in biological material, these techniques become widespread and generate massive data sets [5]. These experiments yield small sequence reads, also called *tags*, which need to be positioned on the genome. For instance, one transcriptomics experiment delivered $\simeq 8$ million different 27

bp tags, which were then mapped back to the genome. Only the tags mapping to a unique genomic location served to predict novel transcribed regions and alternative transcripts [6]. Generally, further analyses concentrate on those tags mapped to a unique genomic location [7].

The goal of tag mapping is to find for each tag the best matching genomic position. The ELAND program, which belongs to the bioinformatic pipeline delivered with the Solexa/Illumina® 1G sequencer, reports first an exact matching location if one is found, and otherwise seeks for locations that differ by 1 or 2 mismatches.

In the vast pattern matching literature, numerous guaranteed algorithms have been described to match exactly or approximately a pattern in a text (*i.e.* a read in a sequence), but only a few have been implemented to process efficiently tens of thousands of patterns [8]. In the context of read mapping, tools must be able to process millions of reads and thus, programs that exploit a precomputed genome index often prove more efficient [9,10,11,12]. Read mapping tools offer possibilities of approximate matching up to a limited number of differences (generally a few mismatches). However, they usually trade off a guaranteed accuracy for efficiency [13,10,11,12].

Another specificity of read mapping applications is that further processing considers only reads mapping to a unique position in the genome [7]. From a statistical viewpoint, exact matching of a 20 bp read is sufficient to identify a unique position in the human genome [14,15]. This implies that, instead of approximately matching full length reads, it may be as adequate to match, *i.e.* read prefixes, exactly. This would allow to keep the 100% accuracy, while still being efficient. Thus, it is desirable to further investigate whether exact set pattern matching algorithms can be adapted to meet the requirements of read mapping. For instance, it remains open whether an efficient pattern matching algorithm able to process huge read sets without indexing the genome exists.

To perform the mapping task, the user chooses either fast BLAST-like similarity search programs (BLAT [16], MEGABLAST [17], or SSAHA [18]), or specialised mapping tools (ELAND, TAGGER [19], RMAP [11], SEQMAP [13], SOAP [10], MAQ [9], BOWTIE [12], and ZOOM [20]). ELAND is probably the most used one [6,3,2]. While mappers were designed to process the huge tag sets output by NGS and allow only a few of substitutions and/or indels, similarity search tools were intended to find local alignments for longer query sequences, but can be twisted to map tags [3,21]. To speed up the search, both categories of tools follow a filtration strategy that eliminates quickly non-matching regions. The filtration usually requires to match exactly or approximately a short piece of the sequence (*e.g.*, BLAT or SEQMAP). All mappers but one [20] use variants of the *PEX* filter (as called in [8]), which consists in splitting the tag in $k + 1$ adjacent pieces, knowing that at least one will match exactly when a maximum of k errors are allowed. Logically to accelerate the filtration step, several of these tools exploit an index of the genome's words of length q (or q-mers) [16,18,19,12][1], which is stored on disk, loaded in memory once before all searches, and requires a computer intensive preprocessing of the genome [12]. The construction of a human

---

[1] As well as the version 2.0 of SOAP.

genome index lasts several hours even on powerful servers [12]. Among mapping tools, ZOOM distinguishes itself with a filtration relying on spaced seeds, *i.e.* matching subsequences instead of pieces [20].

Here we present a computer program MPSCAN[2], short for Multi-Pattern Scan, that is able to locate multiple reads in a single pass through the searched sequence and study its average time complexity (Section 2). In Section 3, we compare MPSCAN with the fastest of BLAST-like tools and mapping programs in terms of speed and scalability on large tag sets, and also evaluate the accuracy of similarity search tools for this task. We conclude by discussing the practical and algorithmical implications of our findings.

2   MPSCAN Algorithm

MPSCAN, short for Multi-Pattern Scan, is a program for set pattern matching: it searches simultaneously in a text for a set of words (*i.e.* tags) on a single computer (no parallelisation, no special hardware). To enable fast matching of large tag sets, we combine a *filtration/verification* approach with a search procedure based on bitwise comparisons, and a compact representation of the tag set. The tags are loaded in memory at the start and indexed in a trie-like structure, while the text is scanned on-the-fly by pieces.

The filtration strategy, which was explored for sets of up to $100,000$ patterns in [22], is the clue of MPSCAN efficiency. Filtration aims at eliminating most positions that cannot match any tag with an easy criterion. Then, verification checks whether the remaining positions truly match a tag. MPSCAN can handle a tag set in which tags differ in length. However, the filtration strategy works with tags of identical length; thus, it creates internally a set in which all tags are cut to the size of the smallest one (call this size l). The verification tests whether a complete tag matches. Filtration has been extensively applied to speed up similarity search algorithms, as in BLAST or BLAT [16]. MPSCAN's criterion relies on the fact that a matching window must share subwords of length q with the tags. Subwords of length q are called *q-mers*.

For verification purposes we index the tag set with a trie [8]. To save space, we prune the trie at nodes where the remaining suffixes can be stored in approximately 512 bytes. The suffixes are sorted for easier access during the verification phase. The pruned trie allows for efficient lookup speed and memory usage with patterns sharing common prefixes, and the remaining suffixes are efficiently packed, without compromising efficiency. Without pruning, the trie alone would result in unacceptably huge memory usage, as a single trie node takes up dozens of bytes in the form of pointers alone.

2.1   Filtration Strategy

Let us explain the filtration scheme with an example. Assume a set of 3 tags of length $l = 8$: $\{P_1, P_2, P_3\} = \{accttggc, gtcttggc, accttcca\}$, and set q to 5. The

---

[2] MPSCAN is freely available for academic users and can be downloaded at `http://www.atgc-montpellier.fr/mpscan`

overlapping 5-mers of each pattern are given in Figure 2. For a text window W of length 8 to match P_1, we need that the subword starting at position i in W matches the ith q-mer of P_1 for all possible i, and conversely. Now, we want to filter out windows that do not match any tag. If the subword starting at position i in W does not match the ith q-mer of neither P_1, P_2, nor P_3, then we are sure W cannot match any of the tags. Thus, our filtration criterion to surely eliminate any non-matching window W is to find if there exists a position i such that the previous condition is true.

From a set of tags, MPSCAN builds a single q-mer generalised pattern (Fig. 2). A generalised pattern allows several symbols to match at a position (like in a PROSITE pattern where a position *e.g.* $[DENQ]$ matches symbols D, E, N, or Q). However, here each q-mer is processed as a single symbol. Then, MPSCAN searches for this generalised pattern in the text with the *Backward Nondeterministic DAWG Matching (BNDM)* algorithm [8], which efficiently uses bit-parallelism. The basic idea of the algorithm is to recognize reversed factors (or substrings) of the pattern when scanning a window backward. When the scanned suffix of a window matches a prefix of the pattern, we store this position as a potential start of the next window. When we reach a point in the backward scanning where the suffix of the window is not a factor of the pattern, we shift the window forward based on the longest recognized prefix of the pattern except for the whole pattern. If no prefix was recognized, the length of the shift is $l - q + 1$. To achieve this efficiently, we initialize during preprocessing a bit vector $B[s]$ for each q-mer s, where the ith bit in the bit vector is one if the q-mer appears in the reversed pattern in position i. During searching the algorithm maintains a state vector E, where the ith bit is one if the scanned q-mers match the pattern starting at position i. When we read a new q-mer s, the state vector is updated as follows:

$$E = (E \ll 1) \ \& \ B[s] \ ,$$

where $\ll$ shifts the bits to the left and $\&$ performs a bitwise and of the two bit vectors. If the first bit in E is one, we have read a prefix of the pattern, and if all the bits in E are zero, the scanned suffix of the window does not match any factor of the pattern. Figure 1 gives the pseudo code for the filtration phase.

2.2 Optimal Average Complexity of MPSCAN

For a single pattern, BNDM has a sublinear average complexity with respect to the text length n; in other words, it does not examine all characters of the text. The combination of the BNDM algorithm with q-mers was first studied in [22], where it was shown sublinear. Here we prove that, if one sets the value of q relatively to the total number of tags r, MPSCAN average time complexity is not only sublinear with respect to n, but optimal. Indeed, the average complexity of the set pattern matching problem is $\Omega(n \log_c(rl)/l)$ (cf. [23]) and we prove:

Theorem 1. *The average time complexity of* MPSCAN *for searching r patterns of size l in a text of length n over an alphabet of size c is* $\mathcal{O}(n \log_c(rl)/l)$ *if* $q = \Theta(\log_c(rl))$.

```
 1: i ← l − q + 1
 2: while i ≤ n − q + 1 do
 3:    j = 1; last ← l − q + 1
 4:    E = B[s_i] {s_i is the i^th q-mer of the scanned sequence}
 5:    while true do
 6:       if first bit in E is one then
 7:          {the scanned window is a prefix of the pattern}
 8:          if j = l − q + 1 then
 9:             verify an occurrence; break
10:          end if
11:          last ← l − q + 1 − j
12:       end if
13:       if E = 0 then
14:          break {the scanned window is not a factor of the pattern}
15:       end if
16:       E ← (E ≪ 1) & B[s_{i−j}] {s_{i−j} is the (i − j)^th q-mer of the scanned sequence}
17:       j ← j + 1
18:    end while
19:    i ← i + last
20: end while
```

Fig. 1. Pseudo code for the filtration phase of MPSCAN

Proof. We want to prove that the average time complexity of MPSCAN for searching r patterns of size l in a text of length n over an alphabet of size c is $\mathcal{O}(n \log_c(rl)/l)$ if $q = \Theta(\log_c(rl))$. Practically, c equals 4 for DNA sequences.
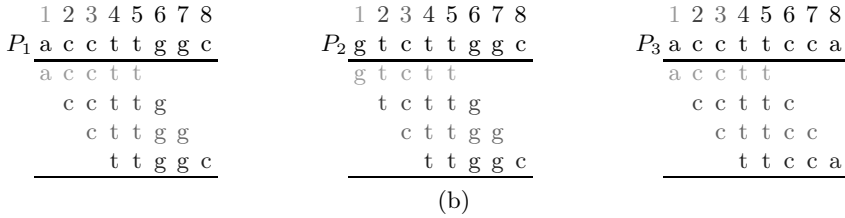
Remember that MPSCAN processes the text in windows and it always reads the windows from right to left. We will call a window *good* if the last q-mer of the window does not match any pattern in any position. All other windows are called *bad*. In a good window, MPSCAN reads only the last q-mer and then shifts the window by $l − q + 1$ characters. In a bad window MPSCAN reads up to l characters and then shifts the window by at least one position (but often more than that).

For the purposes of the proof, the filtering phase of MPSCAN is divided into subphases that we define as follows. Let W_i, $i = 1, 2, \ldots$ be the windows scanned by MPSCAN. The first subphase starts with W_1. Let W_s be the first window of a subphase. Only a good window can end a subphase, but not all of them do. Indeed, the first good window in the series of windows indexed with $i := s + qk$, i.e. W_{s+qk}, with $k = 0, 1, \ldots$ is the last window of that subphase. The next window starts a new subphase. It follows that each subphase consists of X groups of q windows and one good window, with $X \geq 0$ being a random variable. Each of the X groups of q windows starts with a bad window and the rest $q − 1$ windows may be of any type. Figure 3 shows an example of dividing the windows into subphases.

The type of a window following a group of q windows is independent of the first window of the group, because the pattern has been shifted by at least q

$$\{P_1, P_2, P_3\} = \{accttggc, gtcttggc, accttcca\}$$

(a)

```
 1 2 3 4 5 6 7 8              1 2 3 4 5 6 7 8              1 2 3 4 5 6 7 8
P₁ a c c t t g g c          P₂ g t c t t g g c          P₃ a c c t t c c a
   a c c t t                   g t c t t                   a c c t t
     c c t t g                   t c t t g                   c c t t c
       c t t g g                   c t t g g                   c t t c c
         t t g g c                   t t g g c                   t t c c a
```

(b)

$$[acctt, gtctt][ccttg, tcttg, ccttc][cttgg, cttcc][ttggc, ttcca]$$

(c)

Fig. 2. Filtration scheme of MPSCAN. (a) A set of 3 tags of length $l = 8$. (b) The overlapping 5-mers starting at position 1 to 4 (resp. in light, dark, normal, very dark gray) of each tag. (c) The generalised 5-mers pattern for the set of tags.

bad good **good** **good** **bad** bad **bad** good **good** **bad** bad **good** ...
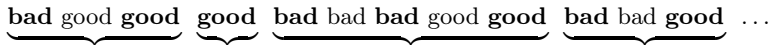
Fig. 3. Dividing the search phase into subphases when $q = 2$. The windows, whose type influences the division, are shown in boldface.

positions between them and the type of a window is determined solely by the last q-mer of the window. If $q \leq l - q + 1$, the type of a window after a good window is also independent of the good window, *i.e.* the q-mer determining the type of the next window contains only characters that have not been previously read. Because each subphase contains at least one good window, the text of length n will surely be covered after $\mathcal{O}(n/(l - q + 1))$ subphases.

The probability that a random q-mer matches any of the patterns in any position is at most rl/c^q, because there are c^q different q-mers and at most rl of these can occur in the patterns (r patterns each of length l). This is also the probability that a window is bad. In a bad window MPSCAN reads the q-mers from right to left. It surely stops when it encounters a q-mer that does not match any q-mer in any of the patterns. In the worst case, MPSCAN reads the whole window and compares it against all the patterns taking $\mathcal{O}(rl)$ time. Note that this is a very pessimistic estimate. In practise, verification is not triggered in all bad windows and even then MPSCAN compares the window against only a few patterns.

In a good window, MPSCAN reads q characters. Therefore in one subphase of filtering, the number of characters read by MPSCAN is less than

$$\mathcal{O}(q) \cdot P(X = 0) + \sum_{i=1}^{\infty} (\mathcal{O}(q) + i \cdot q \cdot \mathcal{O}(rl)) \cdot P(X = i)$$

$$= \mathcal{O}(q) + \sum_{i=1}^{\infty} i \cdot q \cdot \mathcal{O}(rl) \cdot P(X = i) \leq \mathcal{O}(q) + q \cdot \mathcal{O}(rl) \sum_{i=1}^{\infty} i \left(\frac{rl}{c^q}\right)^i.$$

This sum will converge if $rl/c^q < 1$ or equally if $q > \log_c(rl)$ and then

$$\mathcal{O}(q) + q \cdot \mathcal{O}(rl) \sum_{i=1}^{\infty} i \left(\frac{rl}{c^q}\right)^i = \mathcal{O}(q) + q \cdot \mathcal{O}(rl) \frac{\frac{rl}{c^q}}{\left(1 - \frac{rl}{c^q}\right)^2} = \mathcal{O}(q) + q \cdot \mathcal{O}(rl) \frac{rl \cdot c^q}{(c^q - rl)^2}.$$

If we choose $q \geq a \log_c(rl)$, where $a > 1$ is a constant, then $c^q \geq r^a l^a$. Because $a > 1$, $c^q - rl = \Omega(c^q)$ and therefore

$$\frac{1}{c^q - rl} = \mathcal{O}\left(\frac{1}{c^q}\right).$$

Now, the work done by the algorithm in one subphase takes less than

$$\mathcal{O}(q) + q \cdot \mathcal{O}(rl) \frac{rl \cdot c^q}{(c^q - rl)^2} = \mathcal{O}(q) \left(1 + \mathcal{O}\left(\frac{r^2 l^2 c^q}{c^{2q}}\right)\right) = \mathcal{O}\left(q \cdot \frac{r^2 l^2}{c^q}\right) = \mathcal{O}(q)$$

if $a \geq 2$. There are $\mathcal{O}(n/(l - q + 1)) = \mathcal{O}(n/l)$ subphases and the average complexity of one subphase is $\mathcal{O}(q)$. Overall the average case complexity of filtering in MPSCAN is thus

$$\mathcal{O}\left(\frac{n}{l} \cdot q\right) = \mathcal{O}(n \log_c(rl)/m)$$

if $q = a \log_c(rl) \leq l - q + 1$ for a constant $a \geq 2$. The condition $q \leq l - q + 1$ is equivalent to $q \leq (l + 1)/2$. Such a q can be found if $2 \log_c(rl) < (l + 1)/2$ or equally if $r < c^{\frac{1}{4}(l+1)}/l$.   □

The above proof predicts that a good choice for q would be $2 \log_c(rl)$, but in practice a good choice for q seems to be roughly $\log_c(rl)$. If we analysed the complexity of bad windows and verification more carefully, we could bring the theoretical result closer to the practical one.

3   Comparison

The MPSCAN algorithm offers a good theoretical average complexity, but how does it behave in practice and compare to other solutions? We performed search tests to investigate MPSCAN behavior and to compare it to either ultra-fast similarity search tools used for this task (BLAT, MegaBlast, and SSAHA) or to mapping tools. For each tool, we set its parameters to let it search only for exact matches (which is for instance impossible with MAQ). ELAND could not be included in the comparison for we do not have a copy of the program.

Let us first recall some distinguishing features of those similarity search programs. They were designed to search for highly similar sequences faster than BLAST and exploit this high level of similarity to speed up the search. All are heuristic, but are by design better and faster than BLAST for searching exact occurrences of reads in a genome. MegaBlast proceeds like BLAST: it scans

the genome with the query read, which takes time proportional to the genome size. BLAT and SSAHA both use an index of the genome that records the occurrence positions of q-mers in the genome for some length q. Then, they search for all q-mers of the query in the index to determine which regions of the genome likely contain occurrences. This requires a time proportional to the read size. Note that q is the key parameter to balance between sensitivity and speed. Hence, BLAT and SSAHA avoid scanning repeatedly the whole genome, but require to precompute an index.

## 3.1 Speed and Memory with Respect to Text Length

First, we compared the running times of MPSCAN and of all similarity search programs with a set of 200 K-tags and texts of increasing length (Fig. 6, time in log scale). For all programs except BLAT, the running time increases less than linearly with the text length (but BLAT follows the same trend above 50 Mbps). For instance, MPSCAN takes 1.1 sec to search in 10 Mbps of Human chromosome 1, but only 5.6 sec in 247 Mbps: a 5-fold increase of the running time for a 25-fold increase in length. This illustrates well the sublinear time complexity of MPSCAN (Th. 1), which proves to be faster than the reference methods. The behavior is logical: MEGABLAST and MPSCAN first build their search engine, and then scan the text by pieces. The time spent for initialisation of the engine is better amortised with longer texts. This also explains why the memory used by MPSCAN is independent of the text length.

Second, we measured the time and memory footprint needed by MPSCAN and mapping tools to search the complete human genome with one million 27 bp tags. ZOOM requires 17 minutes and 0.9 Gigabytes, RMAP takes 30 min and 0.6 Gb, SEQMAP performs the task in 14 min with 9 Gb, BOWTIE runs in > 6 min with 1.4 Gb and MPSCAN needs < 5 min using 0.3 Gb. MPSCAN runs faster than BOWTIE although the latter uses a precomputed index, and it is three times faster than SEQMAP, the third most efficient tool.

3.2 Scalability with Respect to Number of Patterns

The central issue is the scalability in terms of number of tags. To investigate this issue, we plot their running times when searching for increasing tag sets (Fig 4). The comparison with similarity search tools is shown in Figure 4a. BLAT is by far the slowest tool, while MEGABLAST's time increases sharply due an internal limitation on the maximal number of tags searched at once, which forces it to perform several scans. SSAHA takes full advantage of its index with large pattern sets, and becomes 10 times faster than MEGABLAST. However, MPSCAN runs always faster than BLAT, MEGABLAST, and SSAHA. Especially for more than 400 K-tags, it outperforms other programs by almost an order of magnitude (9.8 s for 700 K-tags instead of 78 for SSAHA, 670 for MEGABLAST and 4, 234 s for BLAT). Importantly, the times needed by other programs increase more sharply with the number of tags than that of MPSCAN, especially after 100K, auguring ill for their scalability beyond a million tags.
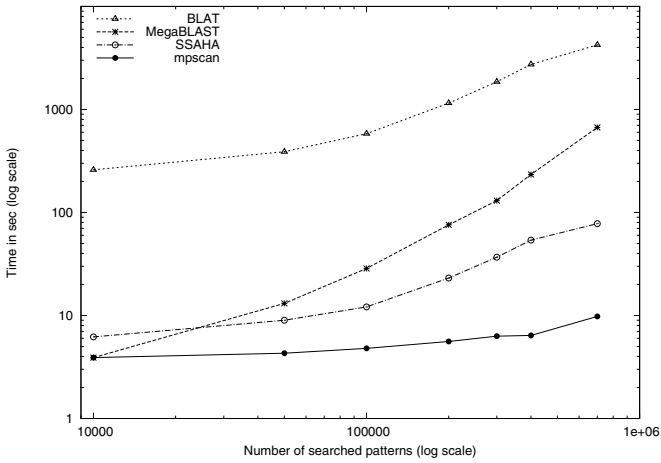
(a)
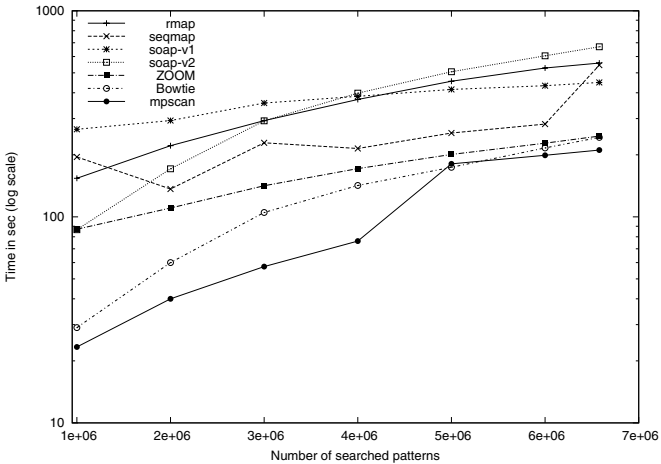
(b)

Fig. 4. Evaluation of scalability. Search times on chromosome 1 (247 Mbp) for increasing tag sets. (a) Comparison with similarity search tools. Search times of BLAT, MegaBlast, SSAHA, mpscan in seconds for 21 bp LongSAGE tags, for sets of 10, 50, 100, 200, 300, 400, and up to 700 Kilo-tags (K-tags). Both axes have logarithmic scales. The curve of mpscan running time is almost flat: for instance doubling the tag set from 200 to 400 K-tags yields a small increase from 5.6 to 6.4 s. Its time increases in a sublinear fashion with the number of tags. For all other tools, the increase of the tag set gives rise to a proportional growth of the running time. *E.g.*, SSAHA needs 23 s for 200 K-tags and 54 s for 400 K-tags. (b) Comparison with mapping tools: Search times of rmap, seqmap, SOAP (v1 & v2), ZOOM, Bowtie and mpscan in seconds (log scale) for increasing subsets of 27 bp ChIP-seq tags. All tools behave similarly and offer acceptable scalability. mpscan remains the most efficient of all and can be 10 times faster than tools like seqmap or rmap. The times do not include the index construction time.
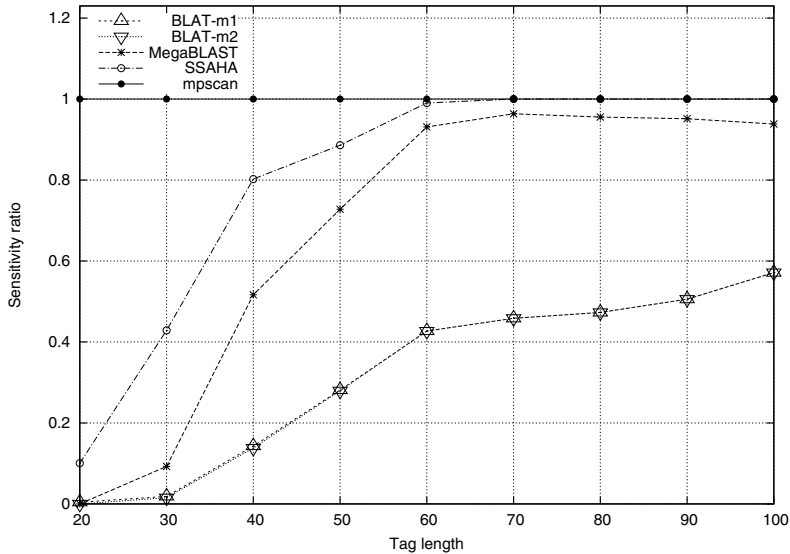
Fig. 5. Sensitivity of MEGABLAST, BLAT, and SSAHA compared to MPSCAN as the percentage of found matches after filtering. Influence of tag length on sensitivity (with $r = 10,000$ tags). BLAT-m1 and BLAT-m2 give BLAT's sensitivity when the filtration criterion asks for one or two seed matches, respectively; BLAT-m1 found necessarily more matches than BLAT-m2. However, here their curves are superimposed. The sensitivity of similarity search tools is low (< 0.5) for tags ≤ 30 bp and reaches a maximum for MEGABLAST and SSAHA at ≥ 60 bp.

Beyond that, we consider specialised mapping tools whose behavior is illustrated in Figure 4b. For this, we used 6.5M 27 bp RNA Polymerase II ChIP-seq tags sequenced in an erythroleukemia cell line (HK652, GEO GSM325934) and took increasing subsets every million tags. All tools exhibit a running time that increases linearly with the number of tags: a much better scalability than similarity search tools. Compared to similarity search tools, all mappers behave similarly, probably due to the resemblance of their filtration algorithm.

Both BOWTIE and SOAP-v2 use a Burrows-Wheeler-Transform index with a similar exact matching algorithm, but it benefits more BOWTIE than SOAP-v2, making BOWTIE the faster of mapping tools. This emphasises how much implementation issues influence efficiency. Among non-index based programs, ZOOM exhibits a behavior close to that of BOWTIE above 3M tags, showing that ultrafast running times are not bound to an index. For moderate tag sets (< 4M tags) MPSCAN is two to four times faster than ZOOM, its fastest competitor in this category. Even if MPSCAN's running time increases from 4 to 5M tags due to a multiplication by 5 of the number of matches, it remains the fastest of all tools for exact matching. This shows that exact set pattern matching can be highly efficient even without a genome index and answers the question asked in the introduction. MPSCAN's filtration strategy is logically sensitive to the
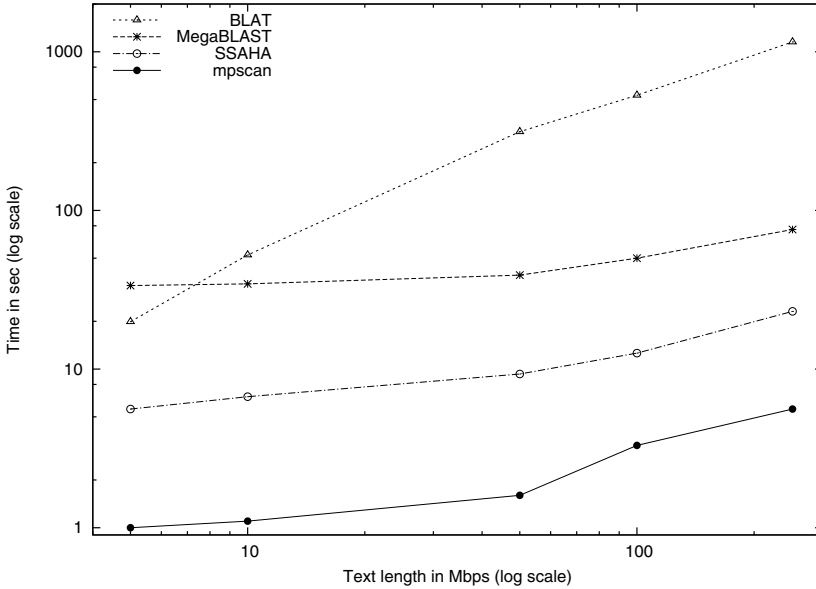
Fig. 6. Search times of BLAT, MegaBlast, SSAHA, mpscan in seconds for 200 Kilo-tags (LongSAGE tags of 21 bp) on increasing pieces of length 5, 10, 50, 100, and 247 Mbp of Human chromosome 1. Both axes have logarithmic scales. These curves illustrate the sublinear increase of time with respect to text length for all tools except BLAT, and the superiority of mpscan in running time.

ratio $\#reads/4^q$, which suggests that using longer computer-words (on 64-bit processors) will improve its efficiency and scalability.

3.3 Accuracy

The mpscan algorithm is guaranteed 100% accurate (and extensive tests have shown that the mpscan program also is): it reports all patterns' occurrences (100% sensitive) and only these (100% selective) [22,8].

Despite the availability of specialised mapping tools, popular heuristic similarity search program like BLAT are still used for read mapping [21], for they can find distant alignments. However to our knowledge, their accuracy has never been assessed in this context. We performed a thorough comparison of their exact matching capacity, since it should be the easiest part of the task. Our results show it is a complex matter: especially their sensitivity is influenced by the numbers of occurrences, the relative length of seeds compared to matches, the parameters set for both building the index and searching.

While all tools (SSAHA, BLAT, and MegaBlast) achieve their best accuracy for long patterns (for ≥ 60 bp, *i.e.* when the seed is guaranteed to fall in each occurrence), all encounter problems finding short patterns (≤ 30 bp). Index and parameters must be adapted to gain sensitivity at the expense of time and

flexibility (one cannot exploit the same index for different tag lengths), which is an issue for digital transcriptomic and ChIP-seq data (≤ 25 bp in [1,3,2]). For instance, with 30 bp patterns, all are less than 50% sensitive with pattern sets $\geq 10,000$ (Fig. 5). For both parameter sets used with BLAT, its sensitivity remains below 0.6 whatever the tag length. The number of tags also has a negative influence on the sensitivity of similarity search tools (data not shown). However, it is logical that similarity search tools have limited accuracy, since they were not designed for exact pattern matching.

The accuracy of mapping tools that allow both exact and approximate matching should be evaluated globally and their dependence to several parameters (tag length, error types, tag number, genome length) should be investigated. Indeed, the underlying definitions of a read best match, the strategies for finding it, as well as the notion of approximation differ among tools, hampering this comparison. This is beyond the scope of this paper. Nevertheless, we have analysed the accuracy of ELAND. Although, we do not have access to the program, some of ELAND's raw results can be downloaded from public repositories like GEO. ELAND searches for exact and approximate matches with ≤ 2 mismatches. We analysed the subset of mapped tags in the ELAND output of the NRSF ChIP-seq data set [2]. ELAND finds only approximate matches for $442,766$ tags, while MPSCAN locates an exact match for $59,571$ of these tags (13% of them).

Such an inaccuracy may impact the final positioning of protein binding or DNA modification sites. This comparison illustrates the difficulty of searching for large tag sets in sequences and the benefit of using a guaranteed pattern matching algorithm for this task.

3.4   Relevance of Exact vs. Approximate Mapping

Currently, new sequencers yield short tags (*i.e.* < 30 bp) in Digital Gene Expression, RNA-Seq, and ChIP-seq experiments (14, 20, and 27 bp in [1,3,6] respectively). Technological developments aim at increasing the tag length to improve the probability of a read to match a unique genomic location. However, the error probability also increases with tag length [21,15]. Altogether, the tag length has an opposite influence on the probabilities of a tag to be mapped and to be mapped at a unique location.

To evaluate the relevance of exact versus approximate matching, we did the following experiment with a Pol-II ChIP-seq set of 34 bp tags (GEO GSM325934). If one maps with MPSCAN the full length tags, 86% remain unmapped and 11% are uniquely mapped. With at most two mismatches, ELAND finds 14% of additional uniquely mapped tags (categories U1 and U2), while mapping the 20 bp prefix of each tag with MPSCAN allows to map 25% of all tags at unique positions (14% more sites than with full length tags).

This result suggests that optimising the final output of a sequence census assay in terms of number of uniquely mapped locations is a complex issue. Approximate mapping is seen as a solution to detect more genomic sites, but it often maps tags at multiple locations [24]. In fine, exact matching may turn out to be a relevant alternative strategy compared to approximate matching. Thus,

the proposed filtration algorithm may be useful in read mapping applications, especially if one considered mapping a substring of the original reads. A more in-depth investigation of this issue is exposed in [15].

4   Discussion

Key biological questions can be investigated at genome scale with new sequencing technologies. Whether in genomic, transcriptomic or epigenomic assays, millions of short sequence reads need first to be mapped on a reference genome. This is a compulsory step in the bioinformatic analysis. We presented an efficient program, MPSCAN, for mapping tags exactly on a genome, evaluated its relevance for read mapping, and compared it to two classes of alternative solutions: i) ultrafast similarity search tools and ii) specifically designed mapping tools. We summarise below some valuable evidence and take-home messages brought by this study.

Similarity search tools are inappropriate for mapping exactly short patterns ≤ 40 bp, since their sensitivity remains too low ($< .5$ for 30 bp long tags). Whatever the number of seeds required to examine a hit, BLAT is the least sensitive among the tested similarity search tools. Its sensitivity never reaches 0.6, even with patterns up to 100 bp. In other words, similarity search tools miss many exact matching locations, which are considered to be the most secure locations in many applications [3,2]. In general, the scalability of similarity search tools is not satisfactory for tag mapping: both the speed of processing and the sensitivity suffer when the number of tags becomes large.

Mapping tools are adequate for this task. They enable the user to map up to millions of tags fast on the human genome, and scale up well. Nevertheless, differences in speed can be important: *e.g.*, an order of magnitude for mapping 2M tags between MPSCAN and SOAP-v2. If most algorithms are similar, from the user viewpoint the programs are not equivalent: neither in flexibility, ease of use, speed, options, nor in accuracy.

From the algorithmic viewpoint, our results suggest that indexing is not required to perform exact mapping of tags on long sequences. In the class of similarity search tools, the superiority in speed of SSAHA compared to BLAT and MEGABLAST is due to its index, but also to its lack of verification, which induces a poor specificity. In our comparison of seven programs (the largest we are aware of), BOWTIE seems the fastest among mapping tools, but never beats the performance of MPSCAN for exact mapping.

ZOOM, which exploits spaced seeds in its filtration scheme, compares favorably in speed to tools using the splitting strategy or PEX filter, such as SEQMAP, RMAP, SOAP. This suggests the superiority of spaced seeds. However, this superiority has a price in terms of flexibility: sets of spaced seeds are specifically designed for a certain tag length and maximum number of mismatches, and different sets corresponding to different parameter combinations are hard coded in ZOOM. For instance, a set of 4 spaced seeds of weight 13 was manually designed to search for 33 bp tags [20]. Hence, adaptation of ZOOM to a new setup re-

quires the design of specific seeds, which is a theoretically hard and practically difficult problem [25,26,27]. The present limitation of ZOOM to patterns up to 64 bp is certainly due to this bottleneck.

In conclusion, we presented an exact set pattern matching program, MPSCAN, which is based on a filtration scheme that had never been applied to read mapping. Our current implementation has pushed the limit on the number of tags by two orders of magnitude compared to previous pattern matching algorithms [8,22]. We conducted thorough comparisons with similarity search algorithms and mapping tools in term of speed and scalability. Our experiments revealed that BLAT-like tools are inadequate for short read mapping both in terms of scalability and of sensitivity, which, to our knowledge, has never been reported before. From the algorithmic viewpoint, we demonstrated the average running time optimality of MPSCAN, which turns out to be very efficient in practice. Compared to mapping tools for exact mapping, MPSCAN runs faster and scales well: it can even compete in efficiency with programs using a sophisticated genome index, like BOWTIE. Since it uses no index, MPSCAN combines flexibility, low memory footprint, and high efficiency, while avoiding a time consuming index precomputation (cf. building times in [12]). Finally, we provide evidence that exact matching approaches can be relevant for read mapping applications, especially in the perspective of longer reads. It remains open to find filtration strategies that achieve efficient "near exact" mapping.

With future generation of sequencers, which promise further increases in sequencing capacity, read mapping may become a bottleneck. Further research in theoretical and practical pattern matching will be needed to tackle this challenging question.

References

1. Kim, J., Porreca, G., Song, L., Greenway, S., Gorham, J., Church, G., Seidman, C., Seidman, J.: Polony Multiplex Analysis of Gene Expression (PMAGE) in Mouse Hypertrophic Cardiomyopathy. Science 316(5830), 1481–1484 (2007)
2. Johnson, D., Mortazavi, A., Myers, R., Wold, B.: Genome-Wide Mapping of in Vivo Protein-DNA Interactions. Science 316(5830), 1497–1502 (2007)
3. Boyle, A.P., Davis, S., Shulha, H.P., Meltzer, P., Margulies, E.H., Weng, Z., Furey, T.S., Crawford, G.E.: High-Resolution Mapping and Characterization of Open Chromatin across the Genome. Cell 132, 311–322 (2008)
4. Schones, D., Zhao, K.: Genome-wide approaches to studying chromatin modifications. Nat. Rev. Genet. 9(3), 179–191 (2008)
5. Mardis, E.R.: ChIP-seq: welcome to the new frontier. Nat. Methods 4(8), 613–614 (2007)
6. Sultan, M., Schulz, M.H., Richard, H., Magen, A., Klingenhoff, A., Scherf, M., Seifert, M., Borodina, T., Soldatov, A., Parkhomchuk, D., Schmidt, D., O'Keeffe, S., Haas, S., Vingron, M., Lehrach, H., Yaspo, M.L.: A Global View of Gene Activity and Alternative Splicing by Deep Sequencing of the Human Transcriptome. Science 321(5891), 956–960 (2008)

7. Barski, A., Cuddapah, S., Cui, K., Roh, T.Y., Schones, D.E., Wang, Z., Wei, G., Chepelev, I., Zhao, K.: High-Resolution Profiling of Histone Methylations in the Human Genome. Cell 129(4), 823–837 (2007)

8. Navarro, G., Raffinot, M.: Flexible Pattern Matching in Strings - Practical on-line search algorithms for texts and biological sequences. Cambridge Univ. Press, Cambridge (2002)

9. Li, H., Ruan, J., Durbin, R.: Mapping short DNA sequencing reads and calling variants using mapping quality scores. Genome Res. 18, 1851–1858 (2008) (in press)

10. Li, R., Li, Y., Kristiansen, K., Wang, J.: SOAP: short oligonucleotide alignment program. Bioinformatics 24(5), 713–714 (2008)

11. Smith, A., Xuan, Z., Zhang, M.: Using quality scores and longer reads improves accuracy of solexa read mapping. BMC Bioinformatics 9(1), 128 (2008)

12. Langmead, B., Trapnell, C., Pop, M., Salzberg, S.: Ultrafast and memory-efficient alignment of short dna sequences to the human genome. Genome Biology 10(3), R25 (2009)

13. Jiang, H., Wong, W.H.: Seqmap: mapping massive amount of oligonucleotides to the genome. Bioinformatics 24(20), 2395–2396 (2008)

14. Saha, S., Sparks, A., Rago, C., Akmaev, V., Wang, C., Vogelstein, B., Kinzler, K., Velculescu, V.: Using the transcriptome to annotate the genome. Nat. Biotech. 20(5), 508–512 (2002)

15. Philippe, N., Boureux, A., Tarhio, J., Bréhélin, L., Commes, T., Rivals, E.: Using reads to annotate the genome: influence of length, background distribution, and sequence errors on prediction capacity. Nucleic Acids Research (2009) doi:10.1093/nar/gkp492

16. Kent, J.W.: BLAT—The BLAST-Like Alignment Tool. Genome Res. 12(4), 656–664 (2002)

17. Zhang, Z., Schwartz, S., Wagner, L., Miller, W.: A greedy algorithm for aligning DNA sequences. J. of Computational Biology 7(1-2), 203–214 (2000)

18. Ning, Z., Cox, A., Mulikin, J.: SSAHA: A Fast Search Method for large DNA Databases. Genome Res. 11, 1725–1729 (2001)

19. Iseli, C., Ambrosini, G., Bucher, P., Jongeneel, C.: Indexing Strategies for Rapid Searches of Short Words in Genome Sequences. PLoS ONE 2(6), e579 (2007)

20. Lin, H., Zhang, Z., Zhang, M.Q., Ma, B., Li, M.: ZOOM! Zillions of oligos mapped. Bioinformatics 24(21), 2431–2437 (2008)

21. Kharchenko, P., Tolstorukov, M.Y., Park, P.J.: Design and analysis of ChIP-seq experiments for DNA-binding proteins. Nat. Biotech. 26(12), 1351–1359 (2008)

22. Salmela, L., Tarhio, J., Kytöjoki, J.: Multipattern string matching with q-grams. ACM Journal of Experimental Algorithmics 11(1) (2006)

23. Navarro, G., Fredriksson, K.: Average complexity of exact and approximate multiple string matching. Theoretical Computer Science 321(2-3), 283–290 (2004)

24. Faulkner, G., Forrest, A., Chalk, A., Schroder, K., Hayashizaki, Y., Carninci, P., Hume, D., Grimmond, S.: A rescue strategy for multimapping short sequence tags refines surveys of transcriptional activity by CAGE. Genomics 91, 281–288 (2008)

25. Kucherov, G., Noé, L., Roytberg, M.: Multiseed Lossless Filtration. IEEE/ACM Transactions on Computational Biology and Bioinformatics 2(1), 51–61 (2005)

26. Ma, B., Li, M.: On the complexity of the spaced seeds. J. of Computer and System Sciences 73(7), 1024–1034 (2007)

27. Nicolas, F., Rivals, E.: Hardness of optimal spaced seed design. J. of Computer and System Sciences 74, 831–849 (2008)