The GEM mapper: fast, accurate and versatile alignment by filtration

Santiago Marco-Sola[1], Michael Sammeth[1], Roderic Guigó[2] & Paolo Ribeca[1,3]

Because of ever-increasing throughput requirements of sequencing data, most existing short-read aligners have been designed to focus on speed at the expense of accuracy. The Genome Multitool (GEM) mapper can leverage string matching by filtration to search the alignment space more efficiently, simultaneously delivering precision (performing fully tunable exhaustive searches that return all existing matches, including gapped ones) and speed (being several times faster than comparable state-of-the-art tools).

In recent years, the superexponential increase in worldwide sequencing capacity[1] has driven a substantial amount of research into the development of efficient algorithms for the analysis of short-read sequence data. In particular, rapid alignment of reads to a genomic reference is often essential. Most mappers have been designed and optimized assuming very short reads (≤100 nt), small deviations from the reference and predominant interest in reads having a unique match or few matches. When a user selects alignment parameters that make the programs deviate from such assumptions, they become considerably less accurate, slower or both. In addition, mappers typically work on a seed-and-extend basis in which a read is aligned by first finding in the reference a short token of the sequence (the 'seed') and then extending the alignment to the rest of the sequence; as few mismatches are usually allowed in the seed, this approach provides good performance but is also inflexible and incapable of returning all existing matches.

As the needs of the field are rapidly shifting, the assumptions described above are now constantly challenged. Some current biological problems (such as nonmodel-organism studies for which matching reference sequences might be incomplete, inaccurate or missing, or cross-species comparisons in evolutionary studies[2]) and new experimental protocols (data in color space, bisulfite-converted sequences or RNA sequencing[3]) require a higher tolerance to errors or more flexible alignment models[4,5]. Other applications (prediction of genomic variation or

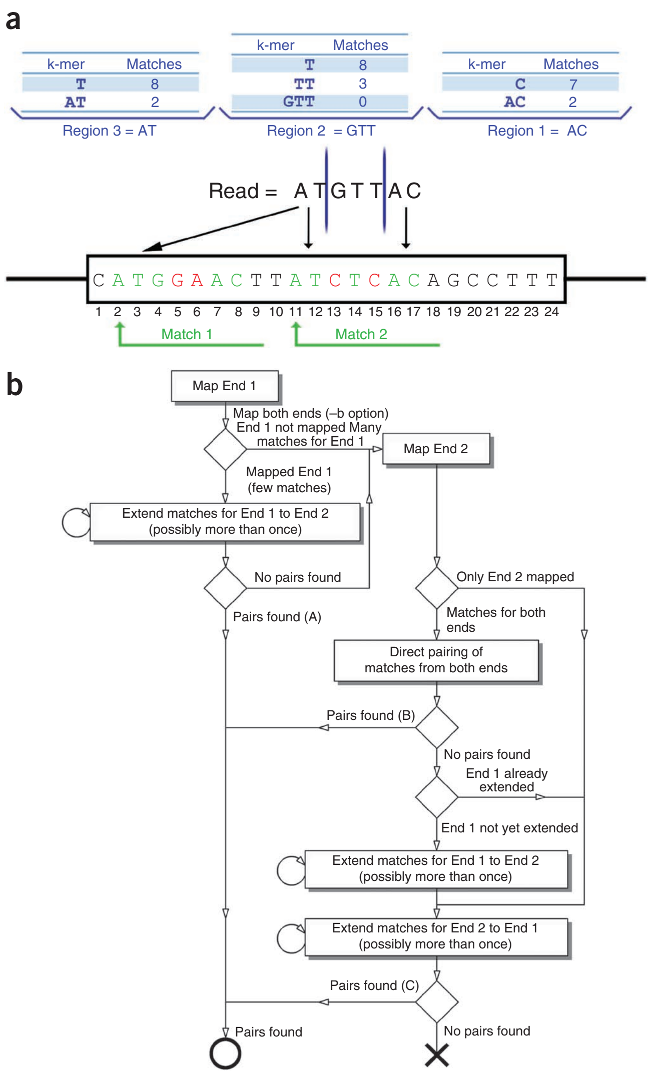

Figure 1 | Salient points of our algorithmic approach. (**a**) An example of region-based adaptive filtering. We aligned the read ATGTTAC to the toy genome CATGGAACTTATCTCACAGCCTTT allowing, at most, two nucleotide substitutions with a 'computational budget' of $t = 3$ (Online Methods). Region 1 (string AC) identifies two candidates at positions 7 and 16, and region 3 (string AT) identifies two candidates at positions 2 and 11. Region 2 (string GTT) does not identify any candidate. The final matches can be found at positions 2 and 11. (**b**) Algorithm for paired-end mapping. If the option '–b' (always map both ends) is set, pairs will be collected from paths B and C (first workflow; Online Methods); if this option is not set, pairs will be collected from paths A and C (second workflow). If the extension parameters are at least as permissive as the alignment parameters, both workflows are guaranteed to find all the paired-end matches such that both ends considered separately map within the specified alignment parameters.

[1]Centro Nacional de Análisis Genómico (CNAG), Barcelona, Spain. [2]Centre de Regulació Genòmica (CRG), Universitat Pompeu Fabra, Barcelona, Spain. [3]Correspondence should be addressed to P.R. (paolo.ribeca@gmail.com).

RECEIVED 29 MAY; ACCEPTED 28 SEPTEMBER; PUBLISHED ONLINE 28 OCTOBER 2012; DOI:10.1038/NMETH.2221

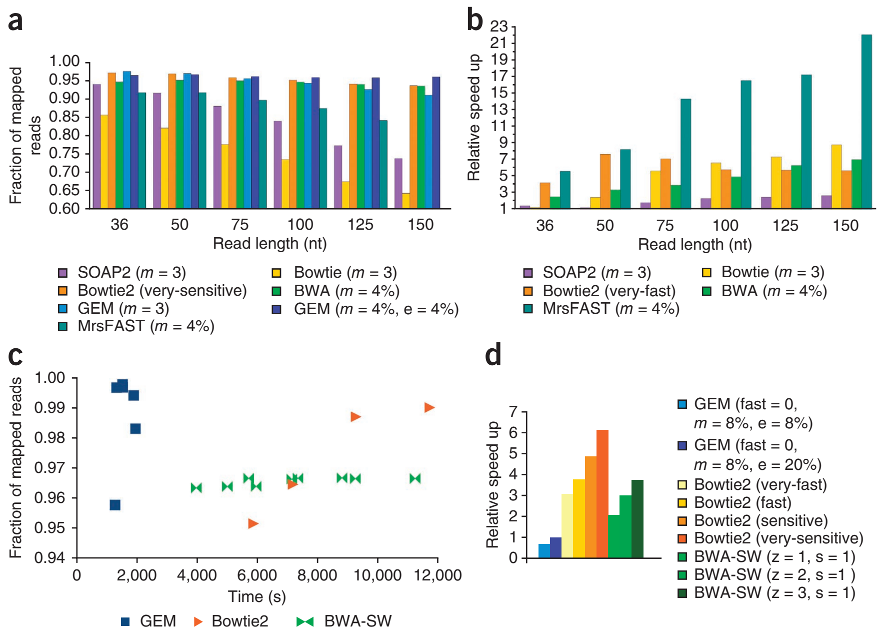

Figure 2 | Benchmarking the GEM mapper on real Illumina GA IIx and Roche 454 sequencing data. (**a**) Fraction of reads mapped by some of the alignment tools and configurations benchmarked in **Supplementary Table 3**, for real GA IIx paired-end reads (36–150 nt). *m*, mismatches (Online Methods). (**b**) Relative speedups of GEM with respect to the mappers benchmarked in **Supplementary Table 3**, for GA IIx reads as in **a** (in each case GEM was run with an alignment setup that matched that of the other mapper as closely as possible). (**c**) Fraction of reads mapped by some of the alignment tools and configurations benchmarked in **Supplementary Table 1** for real 454 single-end reads (572 nt average). BWA-SW, BWA–Smith-Waterman. (**d**) Relative speedups of GEM with respect to the mappers benchmarked in **Supplementary Table 1** for 454 reads as in **c** (in each case, GEM was run with an alignment setup that matched to that of the other mapper as closely as possible; several configurations for each tool were considered). e, fast = 0, fast, very-fast, sensitive, very-sensitive, z = 1, z = 2, z = 3 and s = 1 are command-line parameters (see **Supplementary Protocol**).

metagenomics[6,7]) exploit exhaustive searches of possible matches to increase mapping reliability; in such cases, seed-and-extend alignment is not sufficient.

The concept of 'good alignment' depends on the definition of a string distance between the sequence read and any of its matches in the reference. Mappers usually adopt simple distances enumerating substitutions (Hamming) or edit operations (Levenshtein)[8], but they still present a wide variety of algorithmic setups and scoring schemes. Sometimes such setups are complicated, for example, if their definition can only be understood in terms of the implementation, or even arbitrary, as in the case of the unique alignment modes of several popular mappers such as Bowtie[9] and Burrows-Wheeler Aligner (BWA)[10]; when a read maps multiple times, these mappers break the tie by returning a randomly selected match. Regardless of the definition of a string distance used, however, one can always partition possible alignments to the reference into strata, meaning sets of matches that are all the same distance from the read (**Supplementary Discussion** and **Supplementary Fig. 1**). The strata exploration policy determines the tradeoff between accuracy and computational cost of alignment. Exhaustive alignment algorithms, such as Micro-read Substitution-only Fast Alignment Search Tool (MrsFAST)[7] and GEM, fully explore strata, whereas nonexhaustive algorithms follow a more complicated path, possibly skipping portions of the search space for each read. However, if an exhaustive search is not performed, as in the default alignment models of Bowtie, Bowtie2 (ref. 11) and BWA, it is not possible to make certain statements about the number of matches within a given distance. In particular, it is impossible to decide whether a read is unique or not (this limitation remains true even if the tool offers a probabilistic score to quantify the likelihood of the match).

Here we present our approach to short-read alignment, the GEM mapper. Unlike most other mappers, GEM adopts a filtration-based approach to approximate string matching[12]: all relevant candidate matches are extracted from a Ferragina-Manzini index by suitable pigeonhole-like rules and refined by dynamic programming in bit-compressed representation[13] (Online Methods and **Fig. 1**). This strategy to prune the search space without missing matches, primed with careful optimizations, confers several advantages to the GEM method. First, regardless of alignment parameters, the mapper always performs complete searches: they respect interstrata boundaries and exhaustively find all matches that exist within the search space. Second, the speed of GEM is comparable to or faster than that of several currently used state-of-the-art aligners (**Figs. 2** and **3**); in addition, filtering-based pruning scales well to the range of longer reads targeted by the latest sequencers. Third, because of the flexibility of our algorithmic setup, we implemented an innovative versatile design that allows the user to accurately specify complex alignment models tuned to a specific biological problem.

GEM can also find gapped matches. Not only does GEM report the 'best' matches (those with minimum alignment penalty), it can also explore a tunable number of match strata, effectively resulting in a variable-depth mapping scheme that is adapted to each read (and always outputs separate reliable match counts for each stratum). GEM can perform full paired-end (**Fig. 1b**), quality-aware alignment. Sometimes GEM can replace entire blocks of low-quality nucleotides as wildcards at no additional computational cost, even beyond the nominal number of maximum allowed substitutions, thus typically mapping 70–80% of the Illumina reads containing uncalled bases and passing quality controls. In addition, GEM also implements several special 'hyperfast' modes in which it only aligns reads that, according to a chosen algorithmic criterion, require a small amount of computational resources to be mapped (Online Methods). Notably, such modes differ from the heuristic modes offered by most popular aligners, as the latter can unpredictably miss matches for any read, but all the reads aligned by the GEM fast modes are aligned exhaustively. Empirically, most present-day reads are easily alignable, whereas the reads that are difficult to align are often the same ones for which regular modes would also not find matches; hence, this strategy can map more reads for the same computational budget, as available resources can be concentrated on reads that benefit

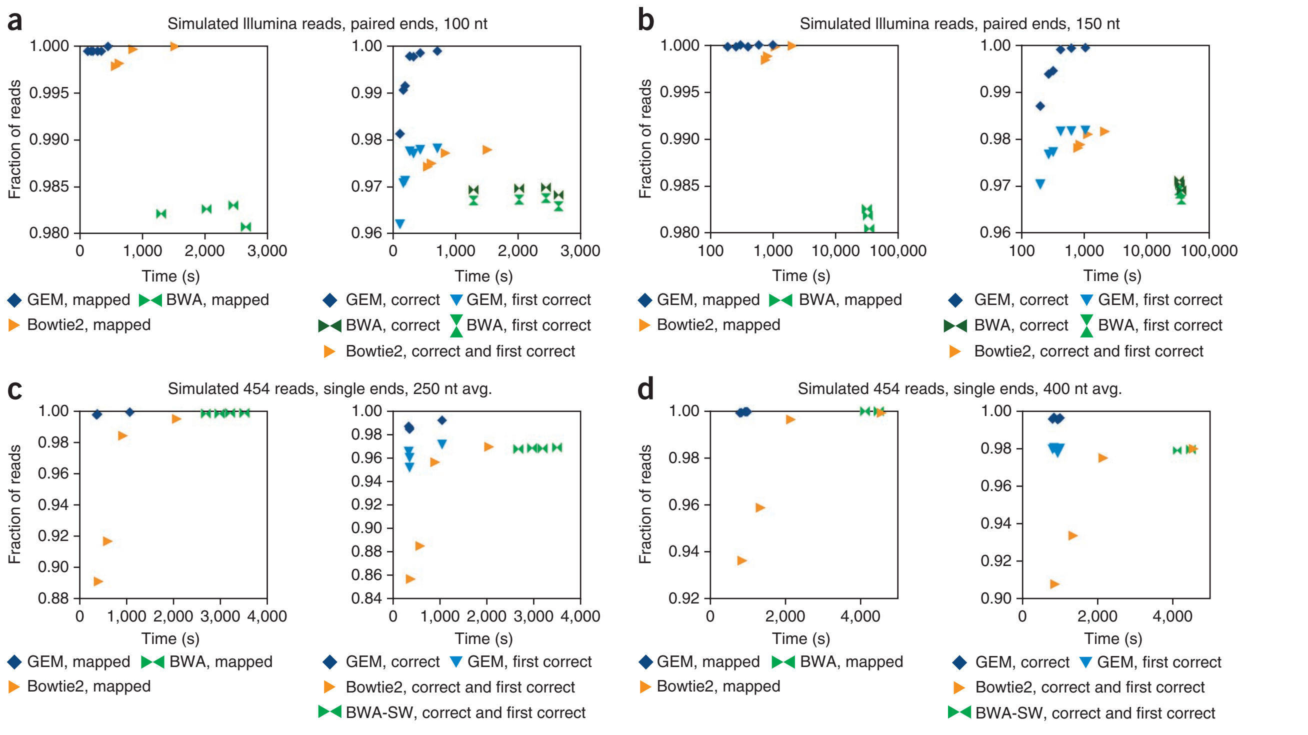

Figure 3 | Several accuracy benchmarks for the GEM mapper on simulated Illumina GA IIx and Roche 454 sequencing data. (**a**–**d**) Fraction of mapped reads as a function of mapping time (left) for several tools and configurations of alignment parameters (each point in the plots corresponds to a line in **Supplementary Table 6** for GA IIx reads and **Supplementary Table 4** for 454 reads). Two measures of accuracy as a function of mapping time (right) for the same tools and configurations shown on the left. Avg., average. The measure 'correct' is the fraction of reads for which the simulated location was correctly retrieved by the mapper as any of the alignments that were output for the read; the measure 'first correct' is the fraction of reads for which the simulated location was correctly retrieved by the mapper as the first (or best) alignment output for the read. When only one alignment per read was reported in the mapping configuration corresponding to a point, the two measures coincide, and this is indicated in the legend. Only a few of the fastest configurations are plotted for each mapper. In **b**, times are plotted on logarithmic scale.

from them. In addition, GEM embeds a fast 'unique-mapping' mode to be used with protocols that discard ambiguous reads (aligning the reads having only one exhaustive match in the reference and flagging the remaining reads as multiply mapping).

Typical results for the GEM mapper on real data from *Homo sapiens* (sets of Illumina GA IIx reads of 36–150 nt and Roche 454 reads with an average length of 572 nt; **Supplementary Protocol**) are reported in **Supplementary Tables 1**–**3** and **Figure 2** together with corresponding benchmarks for several popular aligners. As GEM allows virtually all the relevant alignment parameters to be tuned, we performed fair comparisons by always mirroring the setup of the other mapper (**Supplementary Protocol**). For example, focusing on Illumina reads (**Fig. 2a**,**b**) at the typical length of 100 nt, four general conclusions are apparent. First, GEM is very fast in absolute terms: when using the BWA defaults as alignment parameters, GEM aligned about 40 million single-end reads per central processing unit core per hour on our test machine in paired-end mode. Second, when run with comparable parameters, GEM is typically much faster than other popular mappers. It is 18 times faster than MrsFAST (when run with no insertions-deletions (indels) allowed and all matches in all strata reported). It is six times faster than Bowtie and Bowtie2 run in the low-precision mode that stops after the first match; it takes GEM less time to perform a full search than it takes for Bowtie or Bowtie2 to find a single match. GEM is five times faster than BWA run in its default heuristic mode (and with this choice of parameters, GEM will report all the existing matches, whereas BWA misses some of them). GEM is about two times faster than Short Oligonucleotide Alignment Program 2 (SOAP2)[14] but reports more matches (because of hard-coded limitations on the number of allowed mismatches, SOAP2 is fast but has limited sensitivity). Third, in all considered comparisons, GEM always aligned more reads than its competitors (only BWA and Bowtie2 gave a similar number of results but were also several times slower than GEM). Fourth, the performance of hyperfast mapping modes is noteworthy. When aligning at BWA defaults, the basic fast mode maps only 0.2% fewer reads than does the full mode but runs twice as fast (a more sensitive fast setup, with a Levenshtein distance of 8% instead of 4%, gives a similar speedup and aligns ~1% more reads). We found analogous results with 454 data, for which GEM mapped more reads and was several times faster than its competitors, being about three to five times faster than Bowtie2 and two to four times faster than BWA–Smith-Waterman[15] (**Fig. 2c**,**d**), even though in this test both Bowtie2 and BWA–Smith Waterman, but not GEM, reported only one match per read. In addition, our filtration-based algorithms scale weakly with respect to the read length (behaving much more like the seed-and-extend strategy of Bowtie2 than the exhaustive searches of MrsFAST; **Fig. 2b** and **Supplementary Tables 3** and **4**).

By aligning simulated data sets (generated using the same procedure[10] originally used to evaluate Bowtie2; **Supplementary Protocol**), we also confirmed that the GEM mapper is very accurate in absolute terms and more accurate than other popular aligners (**Fig. 3** and **Supplementary Tables 4–6**). For all considered data sets, GEM recovered as the 'best match' a higher fraction of correct locations, and it typically did so several times faster than its competitors when considering configurations that provide comparable precision. Even when the simulated location was not retrieved as the first match, GEM output it as a subsequent match in virtually all cases.

We argue that the GEM mapper's precision, speed and scaling to higher read lengths are essential for coping with the ever-increasing flood of genomic short-read information and for better problem-driven data analyses, ultimately leading to higher-quality biological insights.

The GEM programs are free for academic noncommercial use and can be downloaded from http://gemlibrary.sourceforge.net.

METHODS

Methods and any associated references are available in the online version of the paper.

Note: Supplementary information is available in the online version of the paper.

ACKNOWLEDGMENTS

This work was supported by grant CSD2007-00050 from the Ministerio de Educación y Ciencia (Spain) and grant 1R01MH090941-01 from the US National Institutes of Health/National Human Genome Research Institute. Additional funding was provided by the European Union 7th Framework integrating project Revolutionary Approaches and Devices for Nucleic Acid Analysis (READNA, funded under grant agreement Health-F4-2008-201418) and the European Union 7th Framework project European Sequencing and Genotyping Infrastructure (ESGI, funded under grant agreement 262055). We thank S. Heath for his thorough revision of the original manuscript. S.M.-S. thanks J. Campos-Laclaustra for his advice. On behalf of the GEM project, P.R. also thanks T. Alioto, J. Camps-Puchades, T. Derrien, S. Djebali, P. Ferreira, I. Gut, S. Heath, D. Gonzalez-Knowles, R. Kofler, V. Lacroix, J. Lagarde and A. Merkel for their continued support and insights.

AUTHOR CONTRIBUTIONS

S.M.-S. designed and implemented algorithms and contributed material to the manuscript. M.S. contributed with fruitful discussions. R.G. initiated the project and contributed with fruitful discussions. P.R. designed and implemented algorithms, was the main architect of the GEM project and wrote the manuscript. All the authors read and approved the manuscript.

COMPETING FINANCIAL INTERESTS

The authors declare competing financial interests; details are available in the online version of the paper.

Published online at http://www.nature.com/doifinder/10.1038/nmeth.2221.
Reprints and permissions information is available online at http://www.nature.com/reprints/index.html.

1. Sboner, A., Mu, X.J., Greenbaum, D., Auerbach, R.K. & Gerstein, M.B. *Genome Biol.* **12**, 125 (2011).
2. Ventura, M. *et al. Genome Res.* **21**, 1640–1649 (2011).
3. Metzker, M.L. *Nat. Rev. Genet.* **11**, 31–46 (2010).
4. Hansen, K.D., Brenner, S.E. & Dudoit, S. *Nucleic Acids Res.* **38**, e131 (2010).
5. Karakoc, E. *et al. Nat. Methods* **9**, 176–178 (2012).
6. Alkan, C. *et al. Nat. Genet.* **41**, 1061–1067 (2009).
7. Hach, F. *et al. Nat. Methods* **7**, 576–577 (2010).
8. Gusfield, D. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology* (Cambridge University Press, 1997).
9. Li, H. & Durbin, R. *Bioinformatics* **25**, 1754–1760 (2009).
10. Langmead, B., Trapnell, C., Pop, M. & Salzberg, S.L. *Genome Biol.* **10**, R25 (2009).
11. Langmead, B. & Salzberg, S.L. *Nat. Methods* **9**, 357–359 (2012).
12. Navarro, G. & Baeza-Yates, R. *J. Discrete Algorithms (Amst.)* **1**, 205–239 (2000).
13. Myers, E.W. *Algorithmica* **12**, 345–374 (1994).
14. Li, R. *et al. Bioinformatics* **25**, 1966–1967 (2009).
15. Li, H. & Durbin, R. *Bioinformatics* **26**, 589–595 (2010).

ONLINE METHODS

Overview. Most popular mappers[16] rely on either hash-table indexing or Ferragina-Manzini indexing[17,18], which is based on the Burrows-Wheeler transform (BWT), as their basic searching scheme. GEM adopts the latter, enabling more flexible searches and having a smaller memory footprint (depending on the parameters used at generation time, the typical size of a GEM index ranges from 3 GB to 6 GB for *H. sapiens*). Being careful enough, however, one is usually able to craft implementations that are largely independent of the basic indexing scheme used. Conversely, when trying to align with errors a sequence read to a large reference, the match space that needs to be explored can be huge, in particular if the alignment model is complex (for example, when it includes gaps or more than one stratum); it therefore becomes essential to specify a good strategy to prune the search. Hence, the major determinant of the final accuracy and performance of an alignment algorithm is usually the high-level pruning strategy used rather than the basic indexing scheme. In particular, some pruning strategies (such as the seed-and-extend strategy) can potentially miss matches for each read, whereas others do not have this limitation.

Here we briefly describe the key principles underlying our approach to effective search pruning. Our approach is based on the combination of two ideas: filtering-based approximate string matching[13] and a powerful method to determine filtering segments (previously unpublished, to the best of our knowledge) that we call region-based adaptive filtering.

We note that our current implementation is not based on the bidirectional BWT: rather than indexing both the sequence of the reference and its reverse, as the vast majority of existing BWT-based aligners do[9–11,14], we only store the sequence alone. This indicates that, although helpful, the bidirectional BWT is not necessary to attain excellent mapping performance.

Filtering-based approximate string matching. In its basic incarnation, filtering is a simple strategy. It consists of a few steps, as follows: (i) The query $\mathbf{Q}$: = $Q_1 \ldots Q_n$ is divided into s nonoverlapping segments. (ii) For each segment s_i, all the corresponding matches up to some distance d_i are retrieved. (iii) Each match for a segment is verified against the entire query. (iv) The nonredundant matches are identified and reported.

The number of segments s, the way the query is subdivided into segments (the intervals $[Q_{1,L} : Q_{1,R}]$, ..., $[Q_{s,L} : Q_{s,R}]$) and the vector $\mathbf{d}$ of the mismatches or differences to be placed in each segment determine speed and accuracy of the query. The query is said to be exhaustive up to some string distance when the choice of $\mathbf{d}$ guarantees that all the matches up to that distance are found. (For simplicity, in the next discussion we consider a number of substitutions, m, rather than some more complicated string distance, but the results presented here can be extended to more general definitions of local error.)

A common way of setting up an exhaustive filtered query up to m mismatches is by (i) taking segments all having the same length, (ii) considering exactly $m + 1$ of them and (iii) considering as candidates only the exact matches for each segment, or, in other words, taking a vector $\mathbf{d} = (0,\ldots,0)$. With such a choice, the pigeonhole principle[8] guarantees that all the matches up to m mismatches will be automatically found, as it is impossible to distribute m mismatches into $m + 1$ slots without leaving at least one hole (that is, one segment without mismatches). This setup is natural when using Ferragina-Manzini–like indexing, which provides very fast exact searches[18].

It follows from our definitions that the efficiency of the search will crucially depend on the total number of candidate matches one has to examine; in turn, the number of candidates will depend on the chosen interval configuration. For instance, when performing standard exhaustive filtering as described above, the higher the number of mismatches m, the more intervals one needs to consider; if the number of segments is too high with respect to the length of the query, then the segments will be too short, and the number of candidates originating from at least some of them will be prohibitive. In general, depending on the sequence of the query, the number of candidates generated by each segment can be very different.

However, the filtering framework is very flexible, and it is not necessary to limit oneself to standard filtering. For instance, the pigeonhole principle can be extended by allowing searches with more than zero mismatches; all the matches within at most m mismatches from the reference will still be obtained, provided that $\lfloor m/d \rfloor$ mismatches are allowed in each segment ($d = |\mathbf{d}|$ is the number of segments here). If this number is kept low (for example, $0 \leq \lfloor m/d \rfloor \leq 2$), then the empirical cost of the search can be even lower than that of a standard filter without sacrificing the possibility of enumerating all matches. As the generalized pigeonhole principle applies to more complicated local error definitions as well, a filtering scheme lends itself almost equally as well to accommodating indels into searches. In addition, in this scheme, one can efficiently implement variable-depth mapping (and the possibility of specifying a tunable number of strata to be explored in addition to the best one) by using the matches already found to better locate the strata of interest.

Our current GEM implementation relies on several layers of optimization to create a robust and efficient filtering-based framework. At both the design and implementation stages, we put particular emphasis on several requirements, namely (i) always ensuring exactness by enforcing pigeonhole-like rules, (ii) being able to perform variable-depth searches and (iii) being able to climb to an intermediate number of mismatches without excessive slowdowns.

Region-based adaptive filtering. Depending on the length scale considered, each sequencing read can be considered as a concatenation of several subsequences, each one being more or less repetitive, that is, having a larger or smaller number of matches in the genome. Naive setups such as the standard filtering described above, which divides the read into equally sized segments, do not exploit the full power of filtering; some of the segments might yield many candidate matches that need to be checked, thus leading to inefficient alignment. Hence, a way of determining a filtering configuration that can be adapted to each read is needed if one wishes to obtain good results; however, how to do this optimally is an open problem and an actively researched one. In this section we briefly describe our solution.

We first pick up a threshold t, which describes the maximum number of candidates we are willing to consider for each filtering segment (our computational budget). We then start scanning the read backward (that is, from right to left, coherently with our Ferragina-Manzini index setup; however, the method would work equally well if applied to an ordinary left-to-right forward search), adding one character more to the current region each time and noting the number of candidates in the reference that correspond exactly to the string being formed. Each time the number of matches falls below the threshold, we start a new region. An example is illustrated in **Figure 1a**, where t has been set to 3; region 1 initially corresponds to string C (with seven candidates present in the genome) and then is extended to AC (with two candidates). As the number of candidates is now below the threshold, we close the region after AC and restart a new one. Going on with this procedure, we are able to identify three regions in the read (from right to left: AC with two candidates, GTT with zero candidates and AT with two candidates).

Of note, by definition such a procedure guarantees that the number of candidates to be considered per filtering segment will always be less than the threshold; in addition, this algorithm can be implemented straightforwardly in the Ferragina-Manzini indexing framework. However, the method does not give any guarantee about the number of regions that will be identified, and this complicates things if one wishes to implement an exhaustive mapping scheme by enforcing pigeonhole-like rules. For instance, in the example of **Figure 1a**, we obtained three regions. In such a case, one is guaranteed to find all the matches up to two mismatches by just validating the exact matches found for each segment identified. In less favorable cases, however, when the number of regions is less than $m + 1$, one might have to consider as candidates the matches obtained when allowing for more than zero mismatches in each segment, ultimately leading to a computationally more expensive search.

Empirical experience shows that the vast majority of single-end reads are easily alignable according to the method described above, allowing them to be mapped in a fast and exhaustive way. The fast modes described in the main text leave out reads for which the method is unable to identify a sufficient number of regions and align the others.

Match verification and gapped mapping. Once filtering segments are efficiently determined using our adaptive procedure, we verify candidate matches using Myers' fast bit-vector algorithm[19] to align the read against each candidate position in the index. This algorithm uses a bit-compressed representation of a dynamic programming table[20]; less memory is required to store the table, and single steps of the dynamic programming algorithm can be performed over several elements of the table at the same time using a single arithmetic-logic operation.

It should be noted that the alignment parameters specified when filtering and those used during this stage are independent. In particular, when one uses more permissive parameters during match verification—which is often the case, as it allows for long indels to be found—the degree of exhaustiveness of the query is still determined by filtering parameters.

In general, with such a technique, the sum of the lengths of the reported indels will be limited by either the read length (when performing candidate verification) or the maximum allowed insert size (when performing extension followed by alignment).

The GEM output scoring scheme. The set of matches retrieved by the GEM mapper is constrained by the alignment parameters specified on the command line; because our algorithms are exhaustive, they guarantee that all the matches within the specified Hamming or Levenshtein distance will be found. GEM does not internally use, at any time, partial likelihood scores to eliminate alignment branches that seem potentially unproductive. Hence, in the GEM output, we might naturally sort and stratify alignments by increasing Levenshtein distance.

However, because of biological considerations, we adopted a slightly different output scoring scheme. Given the usual assumption that the genome of the organism being resequenced shares a high degree of similarity with that of the reference, the differences observed in short sequence reads can ultimately arise because of either sequencing errors or relatively small genomic variants (SNPs or indels). Consequently, the GEM mapper sorts alignments in the output by assigning to each nucleotide substitution a penalty of +1 (as do most mappers) and to each indel, regardless of its length, a penalty of +1. The rationale for the latter is that, in most cases, it makes sense to consider each short indel as a single 'event' originated by either a rearrangement in the genome or a sequencing error. For instance, in the output of the GEM mapper, an alignment having a single deletion of three bases and an alignment having a single deletion of six bases will seem to belong to the same stratum.

We understand that our choice is quite arbitrary, but we note again that this scoring scheme only influences the sorting of the results in the output and not the nature of the reported alignments. If a different criterion is preferable, the user can easily rescore and resort the matches later by using, for example, other programs of the GEM toolkit (**Supplementary Data**).

Strategies for paired-end mapping. To achieve sensitive and effective paired-end alignment, GEM can refine single-end alignments (obtained by means of filtration, as explained above) through dynamic programming. By 'refinement', we are referring to a combined inter-end gap extension and dynamic programming–based alignment step. In **Figure 1b** we show in more detail the two paired-end mapping workflows currently implemented in the GEM mapper.

The first workflow begins by separately mapping both ends and then pairs the returned single-end matches while respecting the constraints imposed by relative distance and orientation (path B in **Fig. 1b**). If no paired-end match can be found in this way, the program extends the single-end matches previously obtained for either end by dynamic programming using the same Myers algorithm explained above (path C). The second workflow begins by mapping only one end and then extends the matches for the first end to the second end through dynamic programming (path A). If no match is found, the second end is also mapped, and an extension of the matches found to the first end is attempted (path C).

In practice, the implementation of the two workflows is more complicated and corresponds to what is illustrated in **Figure 1b**. Not only are the workflows intertwined, but some optimizations are also performed (for instance, when the first end is very repetitive and many single-end matches are found for it, it is typically faster to map the second end and proceed to direct pairing even if the second workflow is being used).

Under the reasonable assumption that extension parameters are at least as permissive as mapping parameters, both workflows are guaranteed to find all the good-quality alignments for which both ends of the pair lie within the specified mapping parameters.

Such alignments are produced by path B (separate mapping of both ends followed by direct pairing) in the case of the first workflow and by paths A and C (mapping of one end followed by extension to the other end) in the case of the second workflow. In addition, the first workflow can also align (from path C) the reads having only one mappable end, provided that the other end can be found by extending with more permissive parameters the matches for the first one.

Generally speaking, the second workflow requires a proper tuning of the extension parameters; a reasonable estimate of the insert size is needed, which is not available, for instance, when studying structural variations or aligning reads to assembly contigs. Adjusting the number of extensions performed per match to the insert size might also be important to attain better precision (**Supplementary Table 6**). However, under ordinary conditions, the second workflow is faster, as typically one has to single-end map only one of the two ends, and the extension step is subdominant (as rows 19–22 of **Supplementary Table 3** effectively illustrate). Conversely, the first workflow, despite being slower, can be used to retrieve pairs when one of the two ends contains more errors; this workflow might also be preferable when it is convenient to possess separate alignment lists for both ends (for instance, some programs to find structural variations might require this kind of input). Additionally, in some rare situations, such as alignment to very repetitive regions of the genome, it is theoretically possible that direct pairing is faster than extension, and, hence, the first workflow would be more efficient than the second.

Other commodities. Through suitable preprocessing of the input, the GEM mapper can be used to align color-space reads produced by ABI SOLiD platforms[21]. In addition, several ancillary tools are provided to simplify the post-processing of the results (**Supplementary Data**). A first tool transforms files in the GEM output format, performing operations such as rescoring and selection of matches, pipelining of several mapping stages, merging of files and so on. A second tool converts the output of GEM into a PICARD-compliant (http://picard.sourceforge.net) subset of the Sequence Alignment/Map (SAM) format[22]. Despite the ubiquity of SAM, this conversion is left to the user as an optional step because it entails a loss of information (some of the features of the GEM format are not representable in SAM) and the result is bulkier.

16. Li, H. & Homer, N. *Brief. Bioinform.* **11**, 473–483 (2010).
17. Burrows, M. & Wheeler, D.J. *Technical Report 124* (Digital Equipment Corporation, Palo Alto, California, 1994).
18. Ferragina, P. & Manzini, G. in *Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS 2000)* 390–398 (2000).
19. Myers, E.W. *JACM* **46**, 395–415 (1999).
20. Eddy, S.R. *Nat. Biotechnol.* **22**, 909–910 (2004).
21. The Tomato Sequencing Consortium. *Nature* **485**, 653–641 (2012).
22. Li, H. *et al. Bioinformatics* **25**, 2078–2079 (2009).