PaPaRa 2.0: A Vectorized Algorithm for Probabilistic Phylogeny-Aware Alignment Extension

Simon A. Berger and Alexandros Stamatakis

The Exelixis Lab, Scientific Computing Group, Heidelberg Institute for Theoretical Studies,
Schloss-Wolfsbrunnenweg 35, 69118 Heidelberg, Germany
simon.berger@h-its.org;alexandros.stamatakis@h-its.org

Abstract. We present a significantly improved version of our phylogeny-aware alignment extension algorithm PaPaRa for aligning short sequence reads with respect to a given multiple sequence alignment and a corresponding phylogenetic reference tree.
This procedure constitutes a pre-processing step for computing the distribution of short sequence reads on a given reference phylogeny and full-length sequence alignment using maximum likelihood-based phylogenetic placement methods such as EPA or pplacer.
Our algorithm now deploys a novel probabilistic method to more accurately extract, model, and use the indel (insertion/deletion) pattern induced by the multiple reference alignment and corresponding phylogeny. This new approach improves the accuracy of subsequent phylogenetic placements by a factor of two. Also, placement accuracy is now consistently better than for alignments obtained with phylogeny-agnostic methods such as HMMALIGN.
Moreover, we obtained up to 15-fold run time improvements by deploying SIMD (Single Instruction Multiple Data) vector intrinsics to accelerate the alignment kernel.

1 Background

We recently introduced PaPaRa [2] (in the following we refer to the original algorithm as PaPaRa 1.0), a novel method for aligning short sequence reads (query sequences; QS) to an existing reference multiple sequence alignment (RA) and a corresponding phylogenetic reference tree (RT). This algorithm constitutes a pre-processing step for identifying short, anonymous DNA reads (e.g., as obtained from metagenomic samples) based upon a set of known reference organisms/sequences. New sequencing technologies can generate up to several millions of of such short DNA reads with a length ranging between 30 to 450 nucleotides [8]. The large number of reads that can be obtained at low cost allows for using these next-generation sequencing methods for *in-vivo* sampling of microbial communities (e.g., in the human gut [16] or on human hands [6]). For phylogenetic and statistical analyses of the diversity in metagenomic samples, new likelihood-based methods such as the Evolutionary Placement Algorithm (EPA) [1, 15] and pplacer [12] have recently become available. These new placement algorithms help to establish the provenance of the anonymous and diverse environmental sample of short reads by means of assigning the reads to a given —fixed— RT. The reference phylogeny is a fully resolved (strictly bifurcating) *unrooted phylogenetic tree* that is based on the fixed RA of the full length sequences in the RT.

Phylogenetic placement algorithms like EPA or pplacer work by inserting (and removing again) one short read at a time into different edges (branches) of the RT. Thereby, they strive to find the optimal likelihood score of the extended (by one taxon) trees in order to individually and independently determine the best insertion edge (branch) for each read. Evidently, the accuracy of such likelihood-based placements of reads depends on the extended multiple sequence alignment, that entails the RA *and* the QS. Prior to the recent introduction of dedicated QS alignment methods, one could align QS with a RA (containing full length reference sequences) by computing a new multiple sequence alignment (MSA) from scratch based on all sequences (RA *and* QS) using MAFFT [9] or PRANK$_{+F}$ [11] for instance. Because of the extremely large and continuously growing number of QS, this de-novo alignment approach becomes computationally prohibitive. Alternatively, one can keep the existing (potentially manually curated) RA unchanged, and only align the QS with respect to this RA, using dedicated QS alignment methods. One such method for aligning QS with respect to a RA is implemented in the HMMER [3] tool-suite. The program HMMALIGN can be used to align QS against a profile Hidden Markov Model (HMM) that is derived from the RA. The profile-HMM is a monolithic representation of the RA, which does not use the evolutionary information encoded in the RT.

We previously demonstrated, that PaPaRa 1.0 increases the accuracy of subsequent EPA placement runs compared to using HMMALIGN. The abstract alignment procedure of PaPaRa 2.0 does not differ significantly from the original implementation. The algorithm derives a set of ancestral state (AS) vectors [2] using the RA and RT. Each AS vector represents a hypothetical common ancestor of the taxa in the RT. Given a RA and RT that contains r taxa, one AS is generated for each of the $2r - 3$ edges in the RT. The AS vectors in PaPaRa 1.0 consisted of two components: (i) the ancestral sequence profile, which is derived from the parsimony state-vectors [7, 14] at each end of the edge; (ii) a tree-derived gap-signal represented as a set of binary flags that are generated along with the parsimony state-vectors using a set of empirical rules (see [2] for details).

The general PaPaRa algorithm has two phases: Initially, the alignment scores for aligning all QS against all AS are calculated. For each QS, the algorithm keeps track of the respective best-scoring AS. This phase is computationally intensive because it requires $O(nr)$ pairwise alignments on a data set with n QS. In the second phase, the algorithm then creates the actual pairwise alignments for each best-scoring QS/AS pair, and writes the original RA and the aligned QS to a PHYLIP-formatted file. The underlying idea is to decompose the RA into two components: (i) a standard DNA alignment to which we apply standard ML and parsimony methods and (ii) a corresponding binary alignment that reflects the gap/non-gap (indel) structure of the alignment at hand. The latter is used to compute the respective gap probabilities using a two-state likelihood model in PaPaRa 2.0.

Te reminder of this paper is organized as follows: In Section 2 we provide an overview of the algorithmic (accuracy improvement) and technical (run-time improvement) innovations in PaPaRa 2.0. In Section 3 we conduct a detailed experimental study to assess the impact of the above algorithmic and technical innovations. We conclude in Section 4.

2 Improvements over the Original Implementation

The new implementation improves upon PaPaRa 1.0 by algorithmic as well as technical innovations. The algorithmic changes improve upon alignment accuracy, while the technical changes substantially improve run-time performance.

At the algorithmic level, we introduced a new, less ad hoc, method to derive the gap signal from the RT and RA, based on a probabilistic model, rather than a set of empirical rules (see [2]). Furthermore, we fundamentally improve the alignment scoring scheme by introducing versatile handling of freeshift gaps. We have now also added the capability to optionally insert gaps into the reference alignment instead of deleting insertions from the QS.

At the technical level, we use a new SSE-based SIMD vectorization, which can speed up the computational kernel of PaPaRa by a factor of 15.

2.1 Probabilistic Gap Model

Methodologically, the new gap model, represents the most substantial change in PaPaRa 2.0. Originally, PaPaRa 1.0 derived a set of gap-flags (OPEN/CGAP) for each site in the ancestral state vectors. The purpose of these flags was to keep track of those sites in the ancestral state vectors that were likely to contain a gap. The gap-flags were calculated along with the ancestral sequence profile according to a set of simple empirical rules [2]. For positions that contained a gap according to these rules, the scoring scheme of the dynamic programming algorithm in PaPaRa was modified, such that, matches were penalized and gap insertions in the QS were favored.

In PaPaRa 2.0 we introduce a probabilistic scheme to model gaps (indels). Here, the AS vectors of each node x are augmented by a set of two probability vectors $L_0^{(x)}$ and $L_1^{(x)}$, which contain the respective probabilities of observing a non-gap or a gap character for each site of the AS. The number of entries in the two gap probability vectors corresponds to the number of columns (sites) in the RA. For the terminal (leaf) nodes of the RT, the values in $L_0^{(x)}$ and $L_1^{(x)}$ are initialized according to the sequence of the corresponding taxon in the RA. For non-gap characters, the elements in $L_0^{(x)}$ and $L_1^{(x)}$ are set to 1.0 and 0.0 respectively. For gap characters, they are set to 0.0 and 1.0. For an inner node

p in the RT, the probability vectors are derived from the two child nodes c_1 and c_2 according to a simple time-reversible likelihood model that is computed recursively with the Felsenstein pruning algorithm [5]. Our model uses a set of transition probabilities $P_{ij}(t)$, where i and j take values 0 and 1 for the non-gap and gap cases. Each P_{ij} is the probability of a state i transforming into j within time t (e.g., $P_{01}(0.1)$ is the probability of a non-gap character turning into a gap in time 0.1). The edge values t are fixed and correspond to the edge lengths in the RT that are estimated using the nucleotide data and standard likelihood model. The transition probability matrix for time t is obtained by $P(t) = e^{Qt}$, where Q is the instantaneous transition rate matrix, that is initialized by the prior probabilities $\pi_{\text{non-gap}}, \pi_{\text{gap}}$ of observing non-gap/gap characters in the RA (for more details about this binary model, see [10]):

$$Q = \begin{vmatrix} -\pi_{\text{gap}} & \pi_{\text{gap}} \\ \pi_{\text{non-gap}} & -\pi_{\text{non-gap}} \end{vmatrix}$$

As described in [5], the probability vectors of a parent node p are calculated from the probability vectors of child nodes c_1 and c_2 and the respective edge lengths t_1 and t_2 according to

$$L_k^{(p)} = \left(\sum_{i \in \{0,1\}} P_{ki}(t_1) L_i \right) \left(\sum_{j \in \{0,1\}} P_{kj}(t_2) L_j \right)$$

As mentioned above, PaPaRa 2.0 works by aligning QS against AS vectors that represent the edges in the RT. The corresponding gap probability vectors of an AS are generated by inserting a temporary virtual root node x in the center of the edge under consideration. The corresponding $L_0^{(x)}$ and $L_1^{(x)}$ are then calculated according to the above equation.

The gap probability vectors are not directly used in the dynamic programming algorithm. Instead of directly modulating the scoring scheme with the gap/non-gap probabilities, we derive a single binary gap-flag per site. The flag at a site is set, if the probability of observing a non-gap character ($L_0^{(x)}$) at that site is smaller than the probability of observing a gap character ($L_1^{(x)}$). Thereby, the integration of the gap signal into the scoring scheme is very similar to PaPaRa 1.0 (see eq.1 in the next section). The main difference is that, the gap-flag is no longer derived according to some empirical rules, but using a probabilistic model. There are two main arguments in favor of 'compressing' the gap probabilities into a binary flag rather than using them directly in the alignment scoring scheme. Firstly, the overall scoring scheme of the alignment algorithm is not based on probabilities, but on empirical values. Thus, we do not see an intuitive approach for directly integrating the gap-probabilities with this ad hoc empirical scoring scheme. Nonetheless, we carried out tests that directly use the gap probabilities to modulate the scoring scheme. However, this approach performed worse with respect to accuracy (results not shown) than the simple flag-based method described here. Secondly, from a run-time performance point of view, using the gap probabilities directly, would require using floating point arithmetic in the score calculations, which are generally slower than integer arithmetic (e.g., on current Intel CPUs throughput of most integer instructions is twice as high as for their corresponding floating point instructions; see http://www.agner.org/optimize/instruction_tables.pdf). Using 16-bit integers for calculating and storing alignment scores also allows for a more efficient SIMD vectorization than, for instance using 32-bit single precision floating point values (see Section 2.4).

2.2 Unified Use of Freeshift Gaps

Previously, PaPaRa 1.0 used distinct approaches for penalizing gaps in the QS and in the RA (i.e., deletions and insertions). Gaps in the QS (deletions) were penalized using affine gap penalties, that is, there are distinct penalties for opening and extending gaps. Gaps on either ends of the QS were not penalized (freeshift gaps). The rationale for using the freeshift penalty is, that the QS are short compared to the RA and are expected to be fully contained within the genomic region (e.g., a single 16S gene) covered by the RA. Thus, gaps on the RA side (insertions) were treated with a constant gap extension penalty, and gaps on either side of the RA were not free. When gaps are added to either ends of the RA, this effectively means that the QS which emits this gap, is not fully contained within the genomic region of the RA. While we initially assumed that this would not occur, in practice, this assumption turned out to be overly restrictive in some real use cases. Therefore, we extended PaPaRa 2.0 to also allow for freeshift gaps on either side of the RA. Thereby, QS that do not entirely fit within the RA are allowed to experience an unpenalized hangover at the boundaries of the RA. We further modified the scoring scheme, such that gaps inserted into the RA are also treated with affine gap penalties. The respective recursive formula for the dynamic programming alignment of an AS A and a QS B is given by equation 1:

$$\begin{aligned}
CG^i &= \begin{cases} -3 & \text{if GAP-flag is set for site } i \\ 0 & \text{otherwise} \end{cases} \\
(GP_O, GP_E) &= (-3, -1) \\
(GP_O^i, GP_E^i) &= \begin{cases} (GP_O, GP_E) & \text{if} \, CG^i = 0 \\ (0, 0) & \text{otherwise} \end{cases} \\
S^{i,j} &= \begin{cases} 2 & \text{if } A^i \text{and} B^j \text{match} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

$$\begin{aligned}
H_I^{i,j} &= max \begin{cases} H^{i,j-1} + GP_O + GP_E \\ H_I^{i,j-1} + GP_E \end{cases} \\
H_D^{i,j} &= max \begin{cases} H^{i-1,j} + GP_O^i + GP_E^i \\ H_D^{i-1,j} + GP_E^i \end{cases}
\end{aligned}$$

$$H^{i,j} = max \begin{cases} H^{i-1,j-1} + S^{i,j} + CG^i \\ H_D^{i,j} \\ H_I^{i,j} \end{cases} \tag{1}$$

The basic structure of the scoring method is similar to the original one described in [2]. The original paper also gives the definition of a match between A^i and B^j. The main difference is the term H_I that accommodates affine gap penalties for insertions (i.e., gaps in the RA), which were previously only allowed for deletions (i.e., gaps in the QS). A more subtle difference is that, for the insertion score calculation (H_I) we now use the fixed gap-open/gap-extension penalties GP_O and GP_E, whereas for deletions (H_E) we use the scores as modulated by the site-specific gap-flags (GP_O^i and GP_E^i). Finally, we transformed the score minimization used in PaPaRa 1.0 to a maximization, such that match scores are positive values, while gap penalties correspond to negative values (i.e., we maximize the score). Using positive values for matches and negative values for mismatches simply yields the scoring scheme more intuitive and easier to interpret.

2.3 Inserting Gaps into the Reference Alignment

To extend the applicability beyond downstream analyses using phylogenetic placement methods (EPA, pplacer), PaPaRa 2.0 also offers an option to insert gaps into the RA instead of deleting insertion characters from the QS. This allows for preserving the complete sequence information of a QS. However, based on the underlying PaPaRa principle, we still align only one QS at a time against the RA and RT. In other words, if two or more QS insert gaps into the RA at the same position, these gaps have to be merged. The merging procedure currently does not aim to correctly align QS characters with respect to each other within the RA gaps (see figure 1). The result of this merging procedure resembles a full multiple sequence alignment consisting of the previous RA and the newly inserted QS. Due to the simplistic nature of the merging procedure, QS characters which are aligned against a gap in the RA (e.g., the QS part highlighted by the gray box in Figure 1) should not be interpreted as being correctly aligned, even if they appear as aligned characters in the output file. When the resulting MSA is used with the EPA, the parts of the QS which are aligned against gaps in the RA will not contribute to the likelihood signal, since gaps are treated as missing data in all standard maximum likelihood implementations. Because of this mathematical property, the QS characters inside these RA gaps are simply 'ignored' by the EPA. When a PaPaRa alignment is used for downstream analysis with the EPA or pplacer, we therefore suggest to switch off the RA gap insertion feature.

2.4 SIMD Vectorization

As described in [2] the PaPaRa algorithm exhibits a high degree of parallelism, because a—typically—large number of independent pair-wise QS and AS alignment calculations need to be performed (for n QS and a RT with m taxa, the

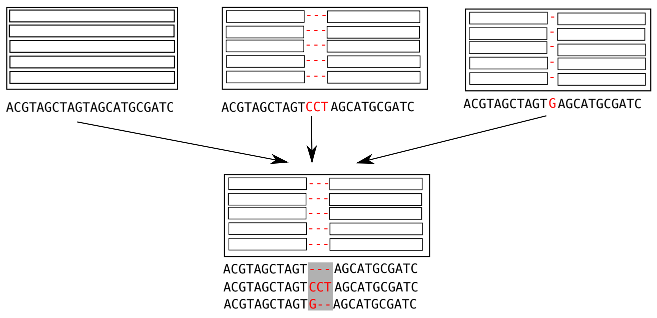
ACGTAGCTAGTAGCATGCGATC
ACGTAGCTAGTCCTAGCATGCGATC
ACGTAGCTAGTGAGCATGCGATC
ACGTAGCTAGT---AGCATGCGATC
ACGTAGCTAGTCCTAGCATGCGATC
ACGTAGCTAGTG--AGCATGCGATC

Fig. 1. Merging overlapping gaps introduced to the RA by different QS

number of independent pair-wise alignments is $O(nm)$). In PaPaRa 1.0 we deployed multi-threading to exploit this coarse-grain parallelism, obtaining good speedups on multi-core architectures. In addition to offering multiple cores, most modern processors also offer SIMD instructions (most commonly SSE on Intel and AMD CPUs) which can be used to exploit data-parallelism at a more fine-grained level.

Using SIMD instructions to accelerate dynamic programming algorithms (e.g., the smith-waterman algorithm), has been extensively studied. See [4][13] for reviews of previous work. Generally, the techniques to exploit SIMD parallelism in pair-wise sequence alignment can be categorized into intra-sequence and inter-sequence approaches. Intra-sequence approaches accelerate the alignment of a single sequence pair (e.g., the swps3 program [4]). The main drawback of this approach is that, when vectorizing the calculation of a single dynamic-programming matrix one has to rely on exploiting wave-front parallelism which limits the degree of parallelism that can be achieved. In contrast to this, inter-sequence vectorization approaches conduct multiple alignments (compute multiple matrices) with a single vector instruction. This is typically achieved by aligning a single sequence against multiple (usually 4, 8, or 16) other sequences at a time. In other words, instead of calculating a single matrix cell $H^{i,j} \in \mathbb{Z}$ per step/instruction, SIMD operations are used to calculate multiple matrix cells $\bar{H}^{i,j} \in \mathbb{Z}^w$, where w is the vector width of the SIMD instructions. The intra-sequence method is particularly efficient if a large number of pair-wise all-against-all alignments need to be conducted, as is the case with PaPaRa.

The actual value w depends on the data type used for the score calculation. In PaPaRa 2.0 we use either 16 bit (DNA data) and 32 bit (protein data) integers with corresponding vector widths of 8 and 4 (i.e., 8x16bit and 4x32bit vectors) respectively. The reason for using different integer types and thus vector widths is that, for greater efficiency, we also use the same data types to store scores as well as the elements of the ancestral state vectors. For protein data, the ancestral state vectors require 20 bits per site, and therefore need to be stored in 32 bit integer values, which correspond to a vector width of 4. For DNA data there are only 4 bits per ancestral state vector site, which can be stored in a 16 bit integer. In this case we also use 16 bit values in the alignment scoring algorithm. For the typically short QS, numeric overflow of these 16-bit integers is currently not an issue.

The vectorized implementation is analogous to the corresponding sequential dynamic programming implementation (Equation 1). The main difference is that, for vector data one can not directly use the standard C++ data types (PaPaRa 2.0 is implemented in C++) and arithmetic operators. Instead, one has to use vector intrinsics. For Intel SSE3 intrinsics, for instance, the `__m128i` data type can either be interpreted as a vector of 8x16bit or 4x32bit integers. It replaces the `short` and `int` data-types and the intrinsic vector operation `_mm_add_epi16` is used instead of the `+` operator. Corresponding intrinsic functions exist for most of the built-in C/C++ data types and arithmetic operations, which yields inter-sequence vectorization relatively straight-forward.

To further improve performance, the vectorized implementation does not keep the whole dynamic-programming matrix in memory. This is because, storing the entire matrix only becomes necessary if the actual alignment shall be generated in the trace-back step. In PaPaRa 2.0 the vectorized algorithm is only used for initial all-against-all score calculations because this step largely dominates execution times. It is therefore sufficient to only store one row of the matrix at a time (see Equation 1: the values in matrix row j only depend on values from rows j and $j-1$. This allows our implementation to overwrite the matrix values from row $j-1$ directly with the newly calculated values in row j). To further increase the cache efficiency of the aligner, we carry out matrix score calculations in a block-based scheme: Instead of calculating the matrix values for an entire row at a time (Figure 2a), only a subset (block) of B columns/entries are calculated (Figure 2b). The block size B defines how many values of the previous matrix row need to be accessed, and is set (depending on the cache size) such as to maintain the referenced values in the L1 cache at all times. The rightmost column of each block is stored and used as starting value for the next block (Figure 2b; columns highlighted in red). An earlier SIMD implementation without this block-based optimization yielded a substantial slowdown when the RA exceeded a certain length (results not shown). The blocking optimization alleviates this limitation.

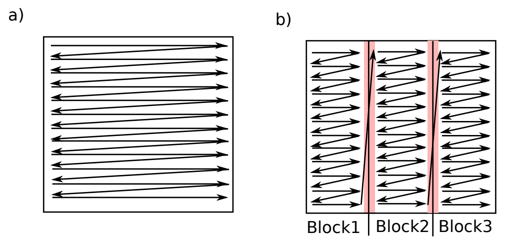

Fig. 2. The order of matrix cell calculations in a standard (a) and block-based (b) inter-sequence dynamic programming implementation. The block-based approach increases data access locality and thereby cache efficiency.

3 Accuracy and Run-Time Performance

3.1 Placement Accuracy

Table 1 provides an accuracy evaluation of PaPaRa 2.0 compared to PaPaRa 1.0 and HMMALIGN (the accuracy values for PaPaRa and HMMALIGN are identical to the results published in [2]). The number in the respective data set name denotes the number of taxa in the respective data set. We restrict our evaluation to the harder case of short QS of length 100 ± 10bp. As in the original evaluation, we provide results for 4 different multiple sequence reference alignments per data set (ORIG, MUSCLE, MAFFT and PRANK$_{+F}$). The measure for quantifying alignment accuracy is exactly the same as in the original evaluation. The values in Table 1 correspond to the RT-based mean distance between the 'true' reference EPA placement and the respective EPA placement after re-alignments with PaPaRa 1.0, PaPaRa 2.0, and HMMALIGN. For a detailed description of the data sets, the method for generating the short QS, and the full evaluation procedure please refer to [2] where we also describe the two distance measures (node-dist (ND) and normalized edge distance (NED%)). The ND represents an absolute distance measure, which does not incorporate the edge lengths of the phylogenetic tree, while the NED% is a relative measure, which normalizes the insertion distance (patristic distance between the calculated and 'true' insertion position) by the corresponding worst-case distance (maximum error) for each placement.

On most data sets PaPaRa 2.0 outperforms PaPaRa 1.0 as well as HMMALIGN. On the largest data set (D1604) and the original alignment (ORIG), the EPA places the PaPaRa 2.0 aligned QS within 0.23 nodes of the reference placement, while for PaPaRa 1.0 and HMMALIGN the distances are 0.28 and 1.31 respectively. On the MUSCLE and MAFFT generated RAs, the performance improvements are similar, while on the PRANK$_{+F}$ aligned RA, PaPaRa 2.0 performs slightly worse than the earlier version (ND 0.44 compared to ND 0.41). However, with respect to the NED% measure, PaPaRa 2.0 consistently outperforms PaPaRa 1.0 as well as HMMALIGN. More importantly, PaPaRa 2.0 no longer shows the comparably large discrepancy between the ND and NED% measure on this data set [2]. On the remaining data sets (except for D628) PaPaRa 2.0 outperforms version 1.0. In contrast to version 1.0 it also consistently outperforms HMMALIGN under both accuracy measures.

3.2 Run-time Performance

To assess the run-time performance of PaPaRa 2.0, we repeated the performance evaluation from [2] where we reported the overall program run times of PaPaRa 1.0 and HMMALIGN on the smallest (D150) and largest (D1604) data set. In contrast to the accuracy evaluation above, the run-time tests are based on 1500 (D150) and 16040 (D1604) QS with lengths 200 ± 60 bp. We used these QS lengths to obtain comparable run times as in the original evaluation. For our experiments we used the same Intel core i5-750 CPU as in the original evaluation,

Table 1. Placement accuracy for the three QS alignment methods under consideration. Results are given using the node-distance (ND) and normalized edge distance (NED%) measures.

		ND			NED%		
	Data Set	PaPaRa	PaPaRa 2.0	HMM	PaPaRa	PaPaRa 2.0	HMM
D150	ORIG	0.52	**0.25**	1.31	1.74%	**1.02%**	1.68%
	MUSCLE	0.57	**0.27**	1.18	2%	**0.99%**	1.51%
	MAFFT	0.58	**0.30**	1.29	1.69%	**1.02%**	1.49%
	PRANK$_{+F}$	0.7	**0.43**	2.1	1.88%	**0.91%**	2.89%
D218	ORIG	2.02	**1.12**	1.78	6.57%	**3.62%**	5.8%
	MUSCLE	1.95	**1.12**	1.73	7.63%	**4.34%**	6.25%
	MAFFT	1.86	**1.13**	1.92	6.21%	**3.73%**	6.14%
	PRANK$_{+F}$	2.04	**1.19**	2.03	7.18%	**4.02%**	6.86%
D500	ORIG	0.57	**0.32**	0.59	1.64%	**0.88%**	1.65%
	MUSCLE	0.6	**0.28**	0.68	1.71%	**0.80%**	1.91%
	MAFFT	0.62	**0.29**	0.69	1.74%	**0.83%**	1.91%
	PRANK$_{+F}$	0.68	**0.35**	0.81	1.88%	**0.96%**	2.22%
D628	ORIG	0.8	**0.77**	1.9	2.07%	**1.85%**	3.89%
	MUSCLE	**1.09**	1.92	3.75	**2.81%**	5.15%	9.38%
	MAFFT	**0.47**	0.55	2.78	**1.11%**	1.13%	5%
	PRANK$_{+F}$	**0.5**	0.58	3.32	1.14%	**1.05%**	5.61%
D714	ORIG	0.55	**0.37**	0.54	1.71%	**0.94%**	1.28%
	MUSCLE	0.5	**0.33**	0.86	1.4%	**0.74%**	1.4%
	MAFFT	0.45	**0.26**	0.75	1.58%	**0.69%**	1.55%
	PRANK$_{+F}$	0.51	**0.30**	1.28	1.47%	**0.59%**	2.48%
D855	ORIG	0.59	**0.53**	1.32	1.03%	**0.63%**	1.67%
	MUSCLE	0.67	**0.44**	1.54	1.22%	**0.71%**	2.32%
	MAFFT	0.66	**0.58**	1.03	1.11%	**0.81%**	1.5%
	PRANK$_{+F}$	0.8	**0.61**	2.28	1.47%	**0.97%**	3.57%
D1604	ORIG	0.28	**0.23**	1.35	0.71%	**0.37%**	1.35%
	MUSCLE	0.43	**0.38**	1.35	0.87%	**0.50%**	1.48%
	MAFFT	0.29	**0.21**	1.21	0.72%	**0.37%**	1.29%
	PRANK$_{+F}$	**0.41**	0.44	2.43	0.95%	**0.58%**	2.41%

with a more recent gcc version (4.6.2). All measurements were carried out using a single CPU core.

For PaPaRa 2.0, the overall program run times range between 27s (D150) and 2590s (D1604). The corresponding run times with the other methods are 345s-39746s (PaPaRa 1.0) and 61s-1030s (HMMALIGN). Note that, the values reported for PaPaRa 1.0 are approximately 10% lower than in [2], while the times for HMMALIGN remain almost constant. We assume that the run time differences are mainly caused by the newer gcc version (presumably the naïve implementation of the alignment core in PaPaRa 1.0 gives the newer gcc version more room for optimization than the well-tuned HMMALIGN kernel). Generally, PaPaRa 2.0 is 12–15 times faster than PaPaRa 1.0 on these data sets. For data sets in this size range, the run time of PaPaRa 2.0 is in the same order of magnitude as HMMALIGN. For larger data sets (in terms of taxa contained in the RA), the program run time of PaPaRa 2.0 increases at a higher rate than for HMMALIGN. Evidently, this is because of the considerably higher time complexity of PaPaRa (both versions) compared to HMMALIGN [2].

The speedup of more than a factor of 12 for the vectorized PaPaRa 2.0 code is larger than expected, given that the SIMD vector width is only 8. Also the alignment kernel of PaPaRa 2.0 is slightly more complex than that of PaPaRa 1.0, because of the affine gap penalties on the RA side, which introduce an additional maximization as well as additions for calculating the $H_I^{i,j}$ values. The unexpectedly large performance gain is most likely due to the revised implementation of the maximization operation and conditional statements of Equation 1. The vectorized implementation deploys a specialized maximization operation (e.g., `_mm_max_epi16`) and bit masks, while PaPaRa 1.0 used explicit branches (i.e., `if/else`). As a result, the inner loop of the vectorized implementation is very compact and, more importantly, contains no conditional jumps.

4 Conclusion and Future Work

We have presented PaPaRa 2.0, an substantially improved and accelerated version of the original proof-of-concept software. The source code of the algorithm is available at http://exelixis-lab.org/software.html. The new version requires less empirical ad hoc rules for extracting information about the indel distribution in the RA by replacing them with a more straight-forward statistical model. This improvement, combined with a better-tuned set of remaining ad hoc alignment scoring parameters improves the overall alignment quality on the datasets used for assessing PaPaRa 1.0. While the new algorithm performs slightly worse than PaPaRa 1.0 on a small number of data sets, it now consistently outperforms HMMALIGN. Performance-wise, the slightly more complex scoring scheme of PaPaRa 2.0 is more than alleviated by the new vectorized alignment core. Despite its high time complexity, PaPaRa 2.0 scales well to large data sets. PaPaRa 2.0 can also deploy multiple threads on multi-core systems. We recently ported the alignment kernel of PaPaRa 1.0 to general purpose graphics processing units (GPUs) using OpenCL. By combining the GPU implementation with a vectorized *and* multi-threaded CPU implementation, we designed a hybrid GPU/CPU system. On a typical desktop system with an Intel core i7-2600 CPU (4 cores) and a NVidia GTX560 GPU, the hybrid approach increases the maximum performance (in terms of cell updates per second) by a factor of 2.4 over the stand-alone vectorized CPU version (technical report available at http://www.exelixis-lab.org/rrdr2012-1.php).

As a more fundamental future improvement, we plan to explore methods that rely on a purely probabilistic representation of the ancestral sequence profiles. The probabilistic gap-signal in PaPaRa 2.0 shows that such a probabilistic representation can increase accuracy. Most importantly, this would allow for removing the ad hoc scoring parameters and potentially yield the algorithm more adaptive to different data sets without any need for parameter tuning. Naturally, the current default parameters in PaPaRa may be biased towards good performance on small subunit rRNA data, which is most abundant among our collection of test data sets. In the tradition of BLAST and other widely used alignment programs, the scoring parameters are user-defined parameters, which can and should be tuned for the application and dataset at hand. Nonetheless, designing a parameter-free, adaptive alignment extension tool represents a desirable research goal to strive for.

References

1. Berger, S.A., , Krompass, D., Stamatakis, A.: Performance, accuracy and web-server for evolutionary placement of short sequence reads under maximum-likelihood. Syst. Biol. 60(3), 291–302 (2011)
2. Berger, S.A., Stamatakis, A.: Aligning short Reads to Reference Alignments and Trees. Bioinformatics (Oxford, England) 27(15), 2068–2075 (2011)
3. Eddy, S.: Profile hidden Markov models. Bioinformatics 14(9), 755–763 (1998)
4. Farrar, M.: Striped Smith-Waterman speeds database searches six times over other SIMD implementations. Bioinformatics 23(2), 156–161 (2007)
5. Felsenstein, J.: Evolutionary trees from DNA sequences: a maximum likelihood approach. J. Mol. Evol. 17, 368–376 (1981)
6. Fierer, N., Hamady, M., Lauber, C., Knight, R.: The influence of sex, handedness, and washing on the diversity of hand surface bacteria. Proceedings of the National Academy of Sciences 105(46), 17994 (2008)
7. Fitch, W., Margoliash, E.: Construction of phylogenetic trees. Science 155(3760), 279–284 (1967)
8. Karow, J.: Survey: Illumina, solid, and 454 gain ground in research labs; most users mull additional purchases (Jan 2010), http://www.genomeweb.com/sequencing/survey-illumina-solid-and-454-gain-ground-research-labs-most-users-mull-addition
9. Katoh, K., Kuma, K., Toh, H., Miyata, T.: MAFFT version 5: improvement in accuracy of multiple sequence alignment. Nucl. Acids Res. 33, 511–518 (2005)
10. Lewis, P.: A likelihood approach to estimating phylogeny from discrete morphological character data. Systematic Biology 50(6), 913–925 (2001)
11. Loytynoja, A., Goldman, N.: Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. Science 320(5883), 1632 (2008)
12. Matsen, F., Kodner, R., Armbrust, E.V.: pplacer: linear time maximum-likelihood and bayesian phylogenetic placement of sequences onto a fixed reference tree. BMC Bioinformatics 11(1), 538 (2010)
13. Rognes, T.: Faster smith-waterman database searches with inter-sequence simd parallelisation. BMC Bioinformatics 12(1), 221 (2011)
14. Sankoff, D.: Minimal mutation trees of sequences. SIAM. J. Appl. Math. 28, 35–42 (1975)
15. Stark, M., Berger, S.A., Stamatakis, A., von Mering, C.: MLTreeMap - accurate Maximum Likelihood placement of environmental DNA sequences into taxonomic and functional reference phylogenies. BMC Genomics 11(1), 461+ (August 2010)
16. Turnbaugh, P., Hamady, M., Yatsunenko, T., Cantarel, B., Duncan, A., Ley, R., Sogin, M., Jones, W., Roe, B., Affourtit, J., et al.: A core gut microbiome in obese and lean twins. Nature 457(7228), 480–484 (2008)