Improved Node and Arc Multiplicity Estimation in de Bruijn Graphs using Approximate Inference in Conditional Random Fields

Aranka Steyaert, Pieter Audenaert, and Jan Fostier

Abstract—In de novo genome assembly using short Illumina reads, the accurate determination of node and arc multiplicities in a de Bruijn graph has a large impact on the quality and contiguity of the assembly. The multiplicity estimates of nodes and arcs guide the cleaning of the de Bruijn graph by identifying spurious nodes and arcs that correspond to sequencing errors. Additionally, they can be used to guide repeat resolution. Here, we model the entire de Bruijn graph and the accompanying read coverage information with a single Conditional Random Field (CRF) model. We show that approximate inference using Loopy Belief Propagation (LBP) on our model improves multiplicity assignment accuracy within feasible runtimes. The order in which messages are passed has a large influence on the speed of LBP convergence. Little theoretical guarantees exist and the conditions for convergence are not easily checked as our CRF model contains higher-order interactions. Therefore, we also present an empirical evaluation of several message passing schemes that may guide future users of LBP on CRFs with higher-order interactions in their choice of message passing scheme.

Index Terms—Belief Propagation, Message Passing Order, Conditional Random Fields, de Bruijn Graphs.

1 Introduction

When analysing microbial genomes, for example in a clinical setting or for surveillance reasons, Illumina sequencing platforms are widely used because of their low cost and high throughput. Illumina reads are characterised by their short length (50-300 nucleotides) and low error rate. When analysing a genome based on Illumina data, the first step is often obtaining a whole genome assembly.

In *de novo* genome assembly, *de Bruijn graphs* play an important role as a data structure to efficiently reveal overlap between reads. We briefly revise the most important concepts and refer to [1], [2], [3] for a more detailed description. Fig. 1 illustrates all concepts introduced here. Each node of a de Bruijn graph corresponds to a unique length-k subsequence of a read, called a k*-mer*. Two nodes u and v are connected by a directed arc if a read contains a $k+1$-mer such that its first k characters correspond to the source node u and the last k characters correspond to the target node v. Consequently, each individual read corresponds to a walk in the de Bruijn graph across overlapping k-mers. Two reads that share at least one k-mer will also share at least one node in their respective walks and hence, overlap between reads is established. For computational efficiency, each linear path over nodes $(n_1, \ldots, n_l)$ with $l \geq 2$ for which the out-degree of nodes n_i for all $1 \leq i < l$ is 1 and the in-degree of nodes n_i for all $1 < i \leq l$ is 1 are contracted into a single node whose sequence of length $k+l-1$ represents the l overlapping k-mers. In other words, all non-branching paths are maximally contracted and the resulting nodes are referred to as *unitigs*. The graph in which all k-mers are contracted into unitigs as much as possible is called a *compacted de Bruijn graph*. In practice, de Bruijn graphs prove more efficient in terms of construction and storage than other methods [2].

If all k-mers and $k+1$-mers of the sequenced genome are present in at least one read (i.e., if there are no coverage gaps), then the genome sequence corresponds to a walk in the de Bruijn graph. The number of times a node (resp. arc) is visited by that walk is called the *multiplicity* of this node (resp. arc). *Repeats* (i.e., subsequences that occur multiple times in the genome) that are longer than the k-mer (resp. $k+1$-mer) length will cause nodes (resp. arcs) to have a multiplicity > 1 as they will be visited by the walk multiple times (e.g. node ATT in Fig. 1). Sequencing errors can result in spurious nodes and arcs in the de Bruijn graph that are not visited by the walk: such nodes and arcs have multiplicity 0 (e.g. node TGTC in Fig. 1). The walk, and hence, also the node and arc multiplicities are a priori unknown. However, multiplicities of nodes can be estimated from the observed k*-mer coverage*, i.e., the number of times a k-mer occurs in the reads. Similarly, the multiplicities of edges can be estimated from the observed $k+1$-mer coverage. The coverage of a node/arc depends on its multiplicity and on the *sequencing depth* of the dataset. The sequencing depth is the average number of times a nucleotide of the genome is covered by a read. Note how, in Fig. 1 the coverage of each node/arc with multiplicity 1 varies around 8.5, while the coverage of nodes/arcs with multiplicity 2 lies closer to 2×8.5. The stochastic relationship between the coverage and multiplicity is usually expressed with a mixture model where each distribution in the mixture corresponds to a certain multiplicity value $(0, 1, 2, \ldots)$ (see the top left of Fig. 1). In theory, if reads are uniformly distributed over the genome, then the k-mer coverage is Poisson dis-

- *A. Steyaert, P. Audenaert and J. Fostier are with IDLab, Ghent University - imec.*
  E-mail: aranka.steyaert@ugent.be, jan.fostier@ugent.be

Manuscript received April 22, 2022; revised XXX.

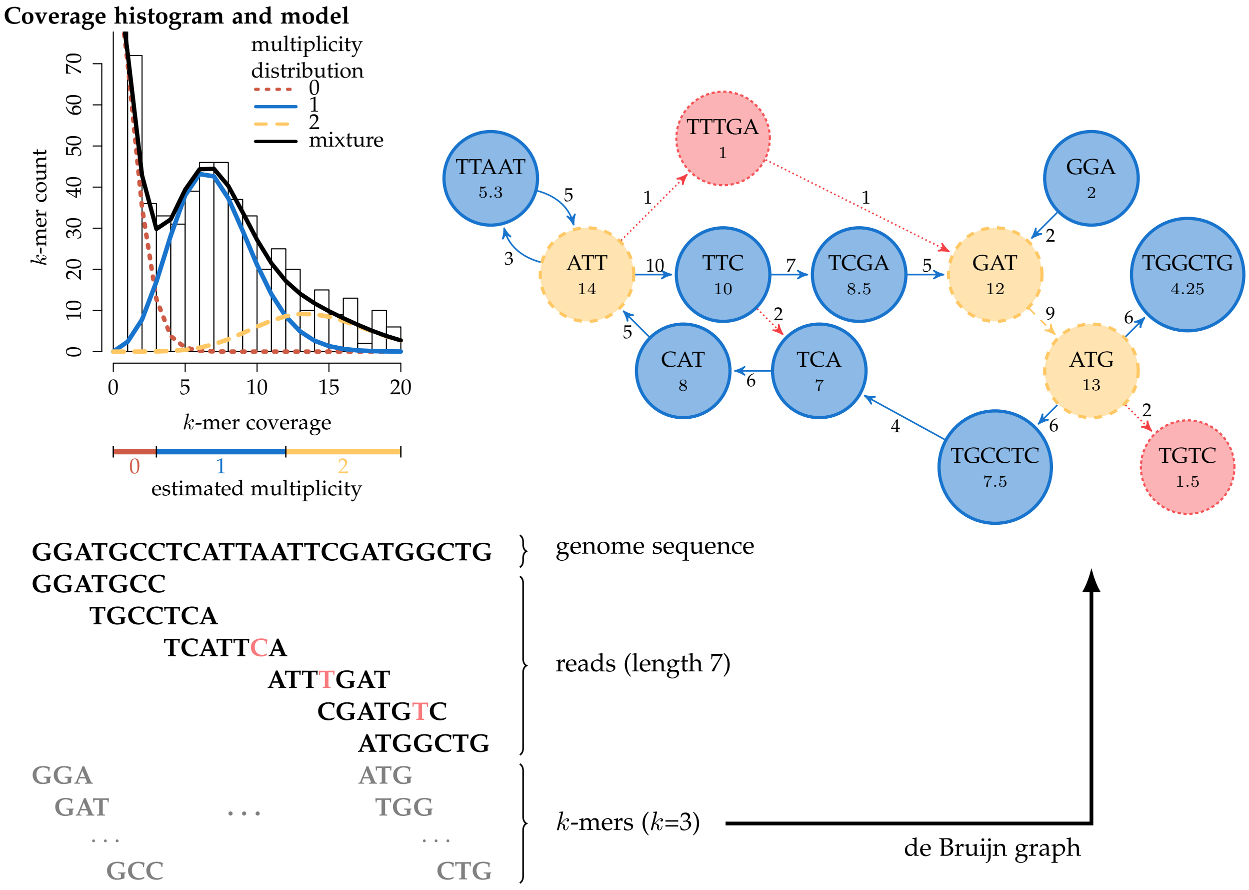


Fig. 1. Illustration of a compacted de Bruijn graph using k-mer size 3 (not all reads are shown). The k-mers are extracted from the reads and form the nodes of the de Bruijn graph. Arcs connect nodes for which the corresponding k-mers are adjacent in a read. Linear paths of connected nodes are contracted into a single node. The number at each node (resp. arc) denotes the k-mer (resp. $k+1$-mer) coverage. For contracted nodes, the average k-mer coverage is used. The original genome is represented as a walk in the graph starting in node GGA. Nodes TTTGA and TGTC, their adjacent arcs and the arc between TTC and TCA will not be used by the walk as they arose from the highlighted sequencing errors in the reads, while ATT, GAT and ATG will be visited twice as these k-mers occur twice in the genome. Nodes and arcs are coloured according to their multiplicity in the genome sequence: sequencing errors (red), unique sequence (blue) and two-fold repeated sequence (yellow). The mixture model representing the relationship between k-mer coverage and multiplicity is shown on the left. The model fit here would assign multiplicity 0 to k-mers with coverage $\in [1,3]$, multiplicity 1 to k-mers with coverage $\in [4,12]$ and multiplicity 2 to k-mers with coverage ≥ 13.

tributed [4]. However, sequencing error bias and coverage bias are sources for overdispersion, which we model with the negative binomial distribution. Additionally, we know that, when all multiplicities are correctly assigned and when there are no coverage gaps, the multiplicity of a node must equal both the sum of the multiplicities of its incoming arcs and the sum of multiplicities of its outgoing arcs. We refer to this property as *conservation of flow of multiplicity*. Note that this property also holds in the presence of sequencing errors, assuming that the corresponding spurious nodes/arcs are correctly assigned multiplicity 0.

Genome assembly tools try to infer (parts of) the walk corresponding to the genome sequence using read and paired-end read information. The accuracy with which node and arc multiplicities are estimated impacts the accuracy and contiguity of the assembly: inferring which nodes and arcs have multiplicity 0 reveals sequencing errors and allows us to remove these nodes and arcs from the graph, while inferring the higher multiplicities ($\geq$ 2) helps us to characterise and resolve the repeats present in the genome sequence. Most existing methods for multiplicity estimation are based on the mixture model fitted to the k-mer spectrum. They use the mixture model as follows: given a node/arc with its observed coverage, the distribution in the mixture with the highest probability of generating that coverage is identified. The node/arc is then assigned the multiplicity that corresponds to that distribution. In practice, this *baseline method* leads to fixed, non-overlapping intervals of coverage values that each correspond to a distinct multiplicity: the lowest coverage values are assumed to correspond to sequencing errors whereas higher coverage value ranges are increasingly assigned higher multiplicity values (see Fig. 1 underneath the coverage histogram). However, biases and errors in the sequencing process can cause a large variability in coverage. Because of this, the distributions in the mixture model often show significant overlap, and therefore, multiplicities cannot be unambiguously assigned using only the observed coverage. This results in many nodes/arcs with erroneous multiplicity assignments. For example, in Fig. 1 node GAT and its outgoing arc would be erroneously assigned multiplicity 1. Similarly, node GGA and its outgoing arc would be assigned multiplicity 0 and thus be labelled as a sequencing error. Nonetheless, this baseline method is often the only information used to determine multiplicities in a variety of methods for de novo genome analysis. This is done for example when estimating genome size and repeat content by tools such as kmerspectrumanalyser [5], CovEst [6], Genomescope 2 [7] and RESPECT [8], that estimate of the number of k-mers with a certain multiplicity in the genome based on a mixture model fit to the k-mer spectrum. Similarly, read error correction methods use the k-mer spectrum to determine a cut-off value between erroneous and true k-mers, although this information is sometimes supplemented by graph topological information from the de Bruijn graph (see [9] for a discussion on error correction methods and their use of the k-mer spectrum and [10] for a discussion on how the errors made by these correctors influence downstream assembly quality). Several genome assemblers such as SGA [11] or ALLPATHS [12] also rely on multiplicity determination using coverage information alone. Some de Bruijn graph based assemblers such as SPAdes [13] and LJA [14] determine multiplicities based on a methodology first described for the EULER-DB assembler [15]: coverage information is used to initially determine multiplicities, but can be combined with network flows on small subgraphs of the de Bruijn graph to propagate estimates with more certainty to lower certainty regions. The hybrid assembler Unicycler [16] uses a de Bruijn graph produced by SPAdes, greedily determines node multiplicities based on coverage and local propagation of multiplicity estimates $= 1$ and combines this information with long reads to improve repeat resolution and assembly.

In [9] we proposed a Conditional Random Field (CRF) model that supplements the coverage mixture model with information provided by the conservation of flow property. Rather than using only the coverage locally observed at a node/arc, coverage information of all nodes and arcs in a small subgraph centered around that node/arc of interest is also taken into account. By imposing conservation of flow at each node, and computing the marginal probability of the node/arc of interest, the assigned multiplicity now depends on the observed coverage of all nodes/arcs within the subgraph. We demonstrated that the CRF model improved multiplicity estimation: it can often correctly assign multiplicity values also to nodes/arcs whose coverage values fall outside of their expected interval [9]. In a subsequent comparison of sequencing error detection accuracy between our method and existing methods, we achieved the highest F1-score on datasets of varying complexity. The results from [9] highlight the importance of incorporating contextual information (e.g. from neighbouring nodes/arcs) when determining multiplicities of k-mers. A similar conclusion was recently drawn by Suzuki and Myers [17], who analyse data from diploid organisms and assign a ploidy-label to each k-mer in a read using coverage information supplemented with contextual information contained in the whole read. While requiring a different k-mer coverage mixture model, the notions of ploidy labels for polyploid organisms and multiplicity in haploid organisms are closely related. To the best of our knowledge Detox [9] and ClassPro [17] are the only methods that incorporate contextual information for each node (resp. k-mer) multiplicity assignment and determine multiplicity 0 together with the other multiplicities. The methods of SPAdes, LJA and Unicycler use heuristics to incorporate some context locally and only do this for a subset of the nodes in the de Bruijn graph. Additionally, they require that a distinction between multiplicity 0 and multiplicity > 0 has already been made by an error correction method.

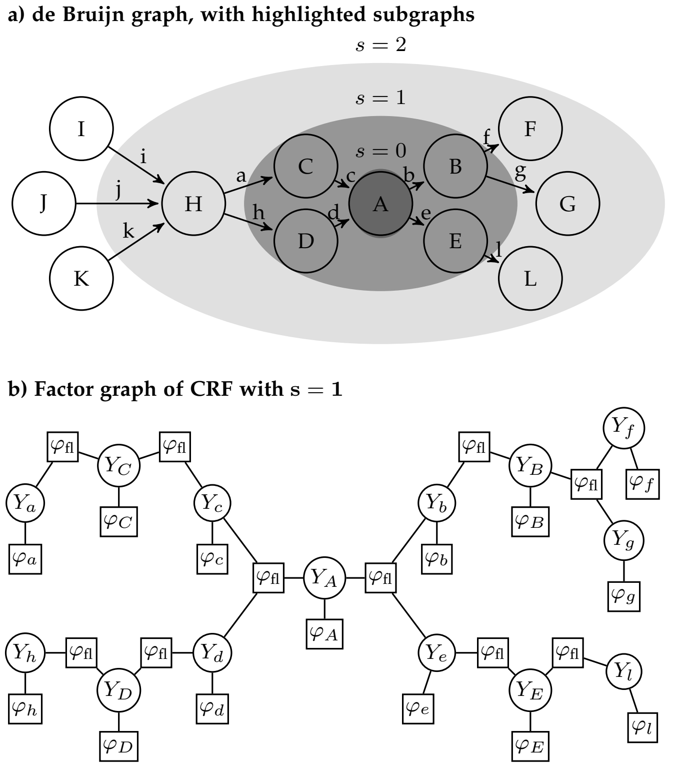
In [9] we relied on exact inference algorithms to compute the marginal probability that a node or arc has a certain multiplicity. However, it is well-known that runtime increases rapidly with increasing number of random variables in the CRF, i.e., with increasing size of the subgraph that is selected. Yet, we also observed that the accuracy of assigned multiplicities increases with increasing size of the subgraph. In this paper, we therefore explore the use of *approximate inference techniques* to determine the multiplicities of nodes and arcs based on one global CRF model for the entire de Bruijn graph. By no longer using subgraphs, coverage information of *all* nodes and *all* arcs in the de Bruijn graph are taken into account when assigning multiplicities. We investigated to which extent such a holistic approach can further boost accuracy.

In order to be widely applicable to many different datasets, we believe it is most beneficial to use Loopy Belief Propagation (LBP) algorithms. Even though little convergence guarantees exist for LBP algorithms, convergence and good estimations of the marginals are often observed even in the cases for which no theoretical guarantees exist [18]. Another popular method of approximate inference in probabilistic graphical models is a sampling based (MCMC) method such as the Metropolis-Hastings algorithm. Sampling-based methods have asymptotic accuracy guarantees for all types of probabilistic graphical models. However, successful sampling-based inference often requires careful design choices for each model and many computations might be needed to obtain a good approximation of the marginals [19]. We believe this makes sampling-based methods less optimal for our application. The successful convergence of an LBP algorithm can depend greatly on the specific order in which messages are propagated. Our CRF model contains higher order interactions (i.e. interactions between more than two variables), and the existing theoretical guarantees are thus not easily translated or applied to our model. Therefore, we analyse different LBP *message-passing orders*. We show empirically under which conditions convergence to a good approximation of the marginals is obtained.

The scope of this paper is thus twofold. On the one hand, we show that modelling the whole de Bruijn graph with a single CRF improves multiplicity assignment accuracy for a range of clinically relevant bacterial datasets, while the use of approximate inference keeps computations feasible. On the other hand, we provide an empirical evaluation of different message passing schemes and their influence on the convergence of approximate inference in the Conditional Random Field models we encounter, and we implement the best performing scheme.

2 CRF MODEL FOR MULTIPLICITY ESTIMATION

A *Conditional Random Field* (CRF) is a probabilistic graphical model often used for classification tasks where a predic-


a) de Bruijn graph, with highlighted subgraphs

b) Factor graph of CRF with s = 1

Fig. 2. Top: de Bruijn graph with highlighted subgraphs of size $s = 0, 1, 2$ centered around node A. Bottom: factor graph representation of the CRF for the subgraph of size $s = 1$ around A. Squares represent factor nodes, while circles represent variable nodes. Each variable i has a singleton factor (φ_i) and from each variable corresponding with a de Bruijn graph node arise two conservation of flow factors (φ_{fl}): one with the incoming arcs and one with the outgoing arcs. Observed variables for each node and arc in the de Bruijn graph are not visualised.

tion can be influenced by contextual (i.e., 'neighbouring') information. It is a popular method for Named Entity Recognition and Image Segmentation tasks [20]. In the first instance, one tries to assign a label (e.g. location, person, time expression or organisation or in the biomedical domain: gene, protein, drug or disease) to each named entity in a text. The probability that one named entity gets a certain label can be influenced by the other parts of the phrase it is contained in and by the labels of other named entities nearby. Moreover, named entities themselves might not be included in a dictionary or reference database and one has to use context alone to determine their label [21]. In Image Segmentation tasks, groups of pixels called segments have to be assigned a label of what they represent in an image. Neighbouring segments are more likely to have the same label, this knowledge is easily translated into a CRF model [19]. Similar to these tasks, we believe it is beneficial to combine the probabilities that each node/arc in the de Bruijn graph has a certain multiplicity based on its coverage (compare with knowledge of a single word or segment) within one probabilistic model such that the final marginal probabilities are based on knowledge about the context of the node/arc (compare with a whole phrase or neighbouring pixel segments).

The CRF model is expressed as a joint (conditional) probability over all unknown variables $\mathbf{Y}$ given observed variables $\mathbf{X}$:

$$P(\mathbf{Y}|\mathbf{X}) = \frac{1}{Z(\mathbf{X})} \prod_{i=1}^{M} \varphi_i(\mathbf{D}_i). \tag{1}$$

Here, $Z(\mathbf{X})$ is the partition function, a normalisation constant. Because a CRF is formulated as a conditional probability distribution, we do not have to define dependencies between the observed variables, nor do we have to assume independence between observations. The joint probability distribution $P(\mathbf{Y}|\mathbf{X})$ factorises into M factors φ_i that quantify the dependencies between the variables in the model. Each factor φ_i has a scope $\mathbf{D}_i$ that is a subset of $\mathbf{X} \cup \mathbf{Y}$, but always contains at least one variable from $\mathbf{Y}$ [19]. The factorisation of the CRF (1) can be uniquely represented with a *factor graph* [18]. This is a bipartite graph that represents the factorised product of (1) as follows [22]: one type of nodes represents the variables in the CRF, while the other type of nodes represents the factors φ_i in the product. Each factor-node is connected with all variable-nodes corresponding to the variables in its scope (see Fig. 2). This graph is not to be confused with the de Bruijn graph itself.

We use the CRF model to predict node/arc multiplicities in a de Bruijn graph as follows. Each node/arc is assigned a variable Y_i that represents its (unknown) multiplicity and variable X_i that represents its (observed) coverage. There are two types of factors. The first type, the *singleton factors* $\varphi_i(Y_i|X_i)$ model the relationship between the coverage X_i observed at a specific node/arc and its unknown multiplicity Y_i. This relationship is modeled using the mixture model of negative binomial distributions described earlier. E.g. the mixture model in Fig. 1 is a combination of three Negative Binomials, the one representing multiplicity 0 has mean 1.0 and variance 1.1, while the ones representing the other multiplicities m have mean $m\lambda$ ($\lambda = 7.5$) and variance $1.1m\lambda$, the weights in the mixture are 200, 300 and 140 for multiplicities 0, 1 and 2 respectively. Given this model, if node A in Fig. 2 has a coverage $X_A = 13$, then $\varphi_A(Y_A = 1, X_A) = 6.99$, while $\varphi_A(Y_A = 2, X_A) = 12.99$. Note that our singleton factors are no probability distribution yet, the normalisation happens at the level of the joint probability. The second type of factors impose conservation of flow of multiplicity at each node of the de Bruijn graph, i.e., the multiplicity Y_n of a node should equal the sum of the multiplicities of its incoming (resp. outgoing) arcs $Y_{a_1}, Y_{a_2}, \ldots$ The *conservation of flow factors* $\varphi_{\text{fl}}(Y_n, Y_{a_1}, Y_{a_2}, \ldots)$ assign a low probability when conservation of flow is violated and a high probability when it holds. These probabilities are determined by a parameter we call the *conservation of flow strength* $\epsilon >> 1$. The CRF contains two conservation of flow factors for each node in the de Bruijn graph: one for its incoming arcs and one for its outgoing arcs. In Fig. 2 $\varphi_{\text{fl}}(Y_A = 1, Y_b = 1, Y_e = 0) = \varphi_{\text{fl}}(Y_A = 2, Y_b = 1, Y_e = 1) = \epsilon$ because for these values of Y_A, Y_b, Y_e $Y_A = Y_b + Y_e$ such as is required by the conservation of flow property. In contrast $\varphi_{\text{fl}}(Y_A = 1, Y_b = 1, Y_e = 1) = 1$ because these multiplicity assignments violate the conservation of flow property. Based on the conservation of flow factors alone, multiple multiplicity combinations are possible. The final assignments will be influenced by a combination of the prior beliefs in the multiplicity of a single node/arc, quantified by the singleton factors, and the likely combinations of multiplicities, quantified by the conservation of flow factors. In this way, the CRF uses the context of a node/arc when determining its multiplicity. We refer the interested reader to [9] which contains a more elaborate description of the factors used in our model as well as a worked out example of a CRF model for a subgraph of the de Bruijn graph and its improvement of multiplicity assignment accuracy.

2.1 Exact inference: variable elimination

To infer the multiplicity Y_i of a node/arc, we compute the marginal probability distribution $P(Y_i|\mathbf{X}) = \Sigma_{\mathbf{Y}\setminus Y_i} P(\mathbf{Y}|\mathbf{X})$. We thus obtain a categorical distribution of the probabilities of multiplicities for that node/arc. We then simply assign the multiplicity with the highest probability. Note that in order to compute the marginal distribution, the partition function $Z(\mathbf{X})$ does not need to be known, as $P(Y_i|\mathbf{X})$ can simply be normalised afterwards.

For a larger number of variables, the computation of the marginal quickly becomes intractable, as all possible multiplicity combinations of $\mathbf{Y} \setminus Y_i$ need to be enumerated. To reduce the required computations, we consider only a subgraph with size s centered around the node/arc under consideration. This subgraph includes all nodes that are reachable within a distance of at most s nodes from the central node/arc as well as their adjacent arcs (see Fig. 2). For $s = 0$, only the central node/arc itself is taken into consideration and the CRF method degenerates into the baseline method that relies only on the coverage mixture model. Additionally, rather than exhaustively enumerating all possible multiplicity combinations of $\mathbf{Y} \setminus Y_i$, we use the computationally more efficient variable elimination algorithm (VE) [19] to compute the marginal distribution. Nevertheless, for larger values of s (i.e., a large number of variables in the CRF), the use of exact inference techniques becomes computationally impractical.

2.2 Approximate inference: belief propagation

In order to do inference computations on a CRF for the entire de Bruijn graph, one has to resort to approximate inference techniques. We run a belief propagation (BP) scheme until convergence once, after which marginal probabilities for the multiplicities of all nodes and arcs are easily extracted.

The BP algorithm extends exact inference methods to approximate inference methods. Given the factor graph representation of the CRF, the BP algorithm can be understood as follows. Based on the notation of [23], we denote the variable nodes with small letters i, while the factor nodes are denoted with capital letters I. Given a variable node i, its neighbours $Nb(i)$ represent all factors whose scope contains i. Given a factor node I, its neighbours $Nb(I)$ represent all variable nodes corresponding to the variables in the scope of the factor corresponding to I. *Messages* between nodes in the factor graph are then defined by:

$$m_{i \to I}(x_i) = \prod_{J \in Nb(i) \setminus I} m_{J \to i}(x_i), \tag{2}$$

$$m_{I \to i}(x_i) = \sum_{x_I \in Nb(I) \setminus i} \varphi_I(Nb(I)) \prod_{j \in Nb(I) \setminus i} m_{j \to I}(x_j). \tag{3}$$

The summation over $x_I \in Nb(I) \setminus i$ denotes that we sum over all different assignments to the variables in $Nb(I) \setminus i$, while the assignment to i is fixed to x_i.

Inference in a CRF can now be viewed as the passing of messages through its factor graph [24]. Messages are first initialised to 1. After initialisation, messages are passed along edges in the factor graph (i.e. calculated according to (2) and (3)) until convergence. Upon convergence we can calculate beliefs $b_i(y_i)$ as approximations to the exact marginals $P(y_i|X)$ as follows:

$$b_i(y_i) = \frac{1}{Z} \prod_{I \in Nb(i)} m_{I \to i}(y_i),$$

where Z denotes a normalisation constant such that all entries in b_i sum to one. Additionally, the factor beliefs can be calculated by:

$$b_I(y_I) = \frac{1}{Z} \varphi_I(y_I) \prod_{j \in Nb(I)} \prod_{J \in Nb(j) \setminus \{I\}} m_{J \to j}(y_j).$$

Factor beliefs can be used to determine joint probabilities of the variables in the set y_I, optionally still summing out some of the variables.

Whenever the factor graph corresponding to a CRF is a tree, the BP algorithm still results in an exact computation of the marginals. However, the factor graphs corresponding to de Bruijn graph based CRFs almost always contain cycles. In the presence of cycles, messages can still be passed through the loopy graph, and the resulting algorithm is thus called Loopy Belief Propagation (LBP). Messages are passed until convergence is reached, or, if non-convergence occurs, until an alternative criterion is reached. A loopy factor graph can also be transformed into a tree-structured graph such that the BP algorithm is exact. However, a tree-structured transformation of the factor graph requires computations with the same order of complexity as the VE algorithm. The transformation into a tree thus comes at a, possibly very large, computational cost [24].

2.3 Implementation

Our software (called Detox) is implemented in C++11. For this reason, we chose to build on an existing C++ library for Probabilistic Graphical Model inference, libDAI [23]. libDAI only explicitly computes and saves the messages $m_{I \to i}(x_i)$, i.e., those that are passed from factors I to variables i. Combining equations (2) and (3) we obtain the following update equation:

$$m_{I \to i}(x_i) \propto \sum_{x_I \in Nb(I) \setminus i} \varphi_I(Nb(I)) \prod_{j \in Nb(I) \setminus i} \prod_{J \in Nb(j) \setminus I} m_{J \to j}. \tag{4}$$

In all experiments presented here we use the following convergence criterion for LBP:

$$\begin{gathered} \max_{x_i} \|b_i^{(t)}(x_i) - b_i^{(t-1)}(x_i)\| < \varepsilon \\ \& \\ \max_{x_I} \|b_I^{(t)}(x_I) - b_I^{(t-1)}(x_I)\| < \varepsilon, \end{gathered} \tag{5}$$

where (t) denotes the t'th iteration of LBP and b_i and b_I are the variable and factor beliefs respectively. Our implementation is available at https://github.com/biointec/detox.

TABLE 1
Six bacterial datasets used for benchmarking. The number of nodes and node multiplicity information refers to the de Bruijn graph with $k = 21$ of the genome itself (i.e., without sequencing errors).

organism	assembly	dataset	number		node multiplicity	
	accession no.	accession no.	of nodes	mean	sd	max
P. aeruginosa	GCF_000006765	ERR1294862	10863	1.52	0.99	19
E. faecalis	GCF_002208945	SRR5448651	1808	2.05	2.22	25
S. enterica	GCF_004224885	SRR8548649	3541	1.91	2.22	44
M. tuberculosis	GCF_009762675	SRR8186769	9715	2.12	2.71	65
S. aureus	GCF_001018845	SRR1955629	3072	2.44	2.74	24
E. coli	GCA_005037805	SRR7896256	7127	2.38	3.78	49

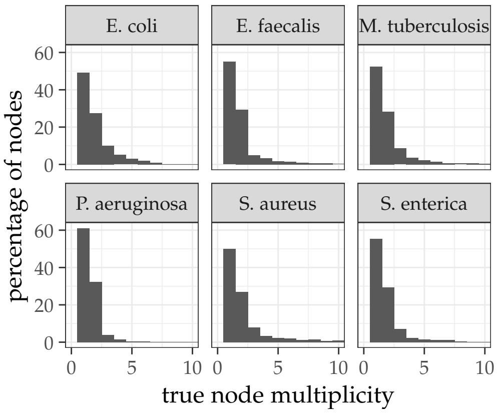


Fig. 3. Histograms of true node multiplicities of error-free compacted de Bruijn graphs from the bacterial datasets used here.

3 Multiplicity determination in de Bruijn graphs: results and analysis

3.1 Datasets

We use six publicly available Illumina datasets (see Table 1) for clinically relevant bacteria to assess the performance of exact and approximate inference algorithms. In all cases, a high-quality assembly is available that matches the read data. All datasets were randomly downsampled to sequencing depths of $10\times$, $15\times$, $20\times$, $25\times$, $30\times$ and $50\times$. This procedure was repeated five times. In total, we thus obtain 180 different read datasets (6 organisms $\times$ 6 sequencing depths $\times$ 5 repeats). Analogously, we generated 180 simulated read datasets using ART [25].

Table 1 provides, for the six organisms, the size of the compacted de Bruijn graph ($k = 21$) in the absence of sequencing errors and coverage gaps and summarizes the true node multiplicities. Even though most of the DNA of microbial organisms is unique (i.e., non-repeated), almost half of the nodes of the de Bruijn graph represent repeated sequences (see Fig. 3). Whereas nodes with multiplicity 1 often contain large sequences, nodes with multiplicity ≥ 2 typically represent relatively short sequences. The associated multiplicities can be as high as 65.

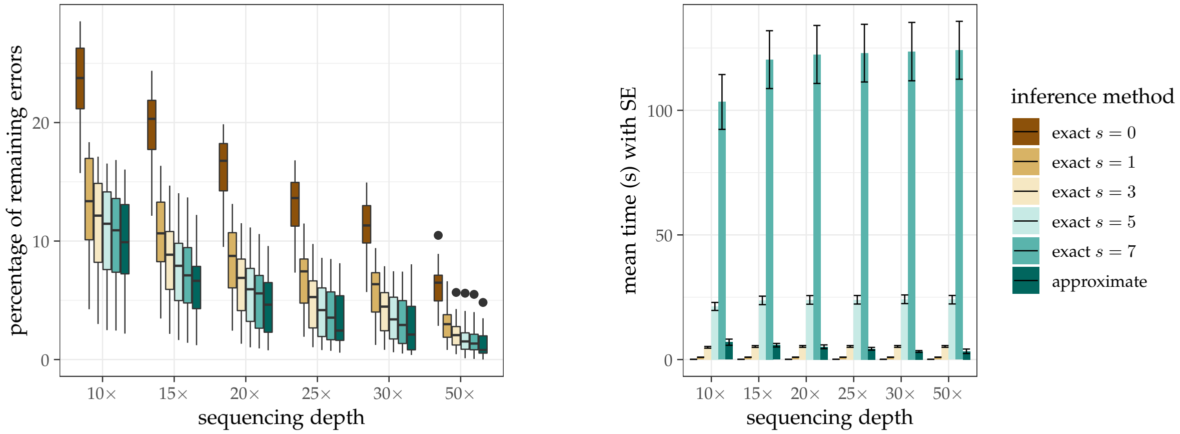
3.2 Workflow

A schematic overview of the workflow of Detox is given in Suppl. Fig. S1. First, a compacted de Bruijn graph ($k = 21$) is constructed from the input reads using BCalm [26]. Our tool consists of three stages. In stage 1, the k-mer (resp. $k + 1$-mer) coverage is counted for all nodes (resp. arcs) in the de Bruijn graph. For unitigs, that consist of multiple k-mers, the average coverage of their constituent k-mers is computed. In stage 2, the model parameters of the mixture model are learned using an expectation-maximization (EM) algorithm. In the E-step, the node and arc multiplicities are inferred (exact inference, subgraph $s = 5$) using the current model parameters. In the M-step, the model parameters are updated based on those multiplicity assignments using the method of moments. Typically, only few (i.e., 10 to 20) iterations are required for the EM algorithm to converge. For computational efficiency, only a subset of nodes and arcs (default: $10\,000$ each) is used to train the model. In stage 3, spurious nodes and arcs that arise due to sequencing errors are removed from the de Bruijn graph using a procedure described in Suppl. Section S2. After graph cleaning and maximally compacting linear paths into unitigs, the resulting de Bruijn graph typically contains a few thousands of nodes and arcs. Most nodes and arcs now correspond to subsequences of the underlying genome. In other words, there are typically only a few sequencing errors left and the graph is a good approximation of the true underlying genome.

3.3 Exact versus approximate inference

Given the cleaned, compacted de Bruijn graph and the fitted mixture model, we determine all multiplicities of nodes and arcs with 1) exact inference (variable elimination algorithm) for different subgraph sizes s (i.e., $s = 0, 1, 3, 5, 7$) and 2) approximate inference (loopy belief propagation) using a single CRF for the entire de Bruijn graph. Based on the results of Section 4.1 we choose 'maximum residual updates with lookahead 0' with weight decay as message passing scheme. Fig. 4 (simulated data) and Fig. 5 (real data) show the percentage of erroneous multiplicity assignments and the average runtime, as a function of sequencing depth. Each box in the box plot summarizes the results across all six organisms and repeated runs.

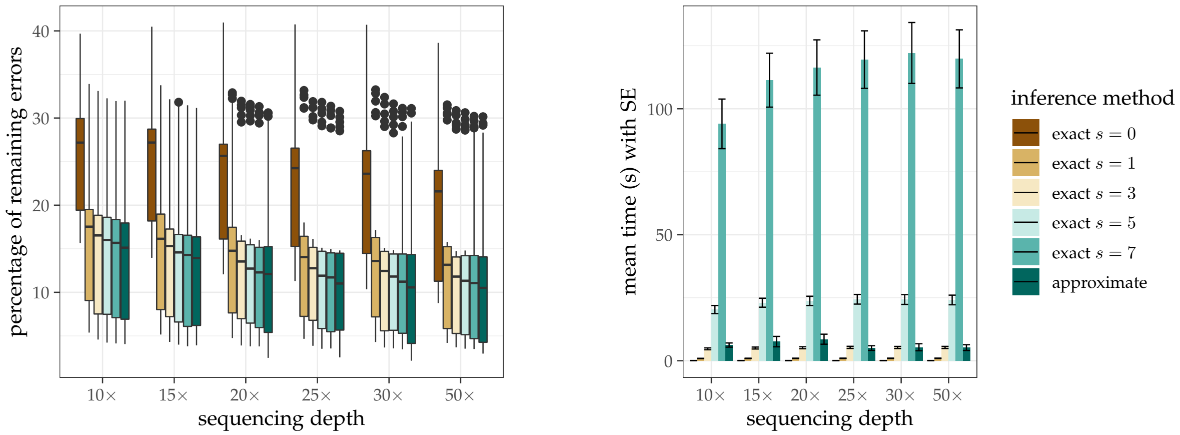
The accuracy with which node multiplicities are estimated increases with increasing size s of the subgraph for exact algorithms and becomes maximal when approximate inference is used, for all sequencing depths and for both simulated and real data (see Fig. 4a and 5a). This confirms the usefulness of the CRF model to infer a node/arc multiplicity not only based on its locally observed coverage, but also on its context, i.e., its surrounding graph topology and the observed coverage of those nodes/arcs. In a certain way, one can think of the approximate inference technique as an asymptotic case where $s = \infty$, i.e., the entire de Bruijn graph is taken into consideration when assigning individual



(a)

(b)

Fig. 4. Comparison of accuracy (left) and runtime (right) of exact and approximate inference methods for different sequencing depths and simulated data. Each result is averaged over 30 datasets (5 simulation runs for 6 different bacterial organisms).



(a)

(b)

Fig. 5. Comparison of accuracy (left) and runtime (right) of exact and approximate inference methods for different sequencing depths and real data. Each result is averaged over 30 datasets (5 downsampling runs for 6 different bacterial organisms).

node/arc multiplicities. Moreover, approximate inference techniques overcome the exponential time complexity of exact inference techniques, which inherently prohibits the use of large subgraph sizes s. This can be observed in Fig. 4b and 5b: approximate inference is faster than exact inference for $s = 5$ and much faster than $s = 7$. Approximate inference successfully allows us to exploit the advantage of a single CRF model for the entire de Bruijn graph within feasible runtime.

With increasing sequencing depth, accuracy increases for all inference techniques. This is because with increasing sequencing depth, the coverage distributions associated with distinct multiplicities overlap less. Therefore, the observed coverage at each node/arc carries more information related to the multiplicity of that node or arc. Nevertheless, the use of the CRF model proves useful even at a sequencing depth of $50\times$. Interestingly, the runtime of the approximate inference technique decreases with increasing sequencing depth. This is due to the fact that a higher sequencing depth is associated with fewer nodes/arcs that have ambiguous coverage, and hence fewer iterations of message passing are required to obtain convergence. Additionally, with higher sequencing depth, there are fewer sequencing errors left in the graph, and fewer coverage gaps (see also Suppl. Table S1).

Even when all available information is combined within a single CRF model, certain multiplicity assignments turn out incorrect, especially at low sequencing depth. One class of errors is related to spurious nodes and arcs in the graph with relatively high coverage: even though these nodes/arcs represent sequencing errors, due to biases in Illumina sequencing technology, their coverage is much higher than expected. This in turn, results in a very low a priori belief that the node/arc represents a sequencing error, which is not reversed by the CRF model. More generally, erroneous assignments may occur for nodes/arcs whose coverage deviates significantly from what is expected by the model. Another class of incorrect assignments relates to nodes/arcs with high multiplicity values. Because the variance of the coverage distribution is proportional to the mean, the negative binomial distributions related to multiplicities m and $m+1$ will increasingly overlap when m becomes larger. Therefore, it is difficult to infer the exact multiplicity. This can also be observed in Suppl. Fig. S7 and S8: more repetitive genomes (i.e., with a larger average node multiplicity, see Table 1) are more difficult to handle.

Often, difficult nodes/arcs with very low or high coverage and/or with high multiplicity values are spatially correlated in the de Bruijn graph. For example, a GC-rich region of the genome may have very low coverage, which manifests itself as a path of connected nodes with low coverage in the de Bruijn graph. In such cases, the multiplicities may not be correctly inferred for this entire subgraph, even when the conservation of flow property holds for most nodes.

The results on real data show a larger percentage of erroneous assignments than simulated data. This is caused by a higher coverage variability and biases that are not reflected well in the simulator used. Additionally, plasmid DNA in real data typically has a much higher abundance than bacterial DNA and hence, its associated node/arc multiplicities will be poorly estimated. Nonetheless, we obtain the highest accuracy within feasible runtime using approximate inference.

Overall, we conclude that the use of a single CRF model and approximate inference techniques for the entire de Bruijn graph is beneficial: significant gains in accuracy are achieved within an acceptable runtime.

3.4 Comparison to other methods

In [9], we compared our multiplicity 0 assignment accuracy to that of state-of-the-art error correction tools that make use of k-mer coverage. We found that we significantly improve error detection. As we further improve multiplicity assignment accuracy in this work, error detection accuracy will also further improve. For this reason we will focus on evaluating multiplicity assignments ≥ 1 here.

To the best of our knowledge, there is no dedicated method for higher order multiplicity determination in de Bruijn graphs, as this remains an internal computation of the assembly pipeline. As developing a complete assembly pipeline is outside the scope of this paper, we compare our multiplicity assignments with those used by several methods for genome size and repeat content estimation. All these methods use what we refer to as the baseline method to determine multiplicity assignments: they fit a mixture model to the k-mer spectrum and determine coverage intervals for each multiplicity. The differences between the methods lie in the probability distributions used in the mixture and in how the erroneous (multiplicity 0) k-mers are modelled. Kmerspectrumanalyzer [5] fits a negative binomial mixture model to the k-mer spectrum, assuming all k-mers with coverage less than $\mu/2$ are erroneous, where μ is the average coverage of the k-mers with multiplicity 1. CovEst [6] fits a Poisson mixture model to the k-mer spectrum combined with a model for the probability that a given k-mer arose from a true k-mer with s point-mutations. Genomescope 2 [7] uses a negative binomial mixture model with only two components (one for the unique k-mers and one for the repeated k-mers). They assign multiplicity 0 to the proportion of low-coverage nodes not explained by the fitted model. Additionally, we compare our multiplicity assignments to those given by Unicycler [16] as intermediary output. Unicycler uses k-mer coverage to estimate multiplicities $=1$ and propagates these estimates to other nodes. In this way it greedily assigns higher order multiplicity based on the topology of the de Bruijn graph. When no estimate was available for a node we used Unicycler's coverage based method (as used to estimate multiplicity $=1$) to provide estimates for all remaining nodes.

We consider 6 different multiplicity bins: one for each multiplicity $\in[0,4]$ and a final bin for multiplicities ≥ 5. Detox and Unicycler output multiplicity estimates for each unitig in the de Bruijn graph. We bin unitigs based on these multiplicity estimates and count all the k-mers for each multiplicity bin using KMC3 [27] ($k=21$). For the other tools, we determine coverage intervals for each multiplicity bin based on their k-mer spectrum model and count the k-mers with coverage in these intervals with KMC3 ($k=21$) (see also Suppl. Section S4). We evaluate the performance of each method using the Jaccard index of the true bins determined by counting k-mers in the reference genome compared with the estimated bins:

$$Jac(m)=\frac{|\text{True}(m)\cap\text{Estimate}(m)|}{|\text{True}(m)\cup\text{Estimate}(m)|},\; m\in\{0,1,2,3,4,5+\}$$
$$Jac=\sum_{m}\frac{Jac(m)}{6}.$$

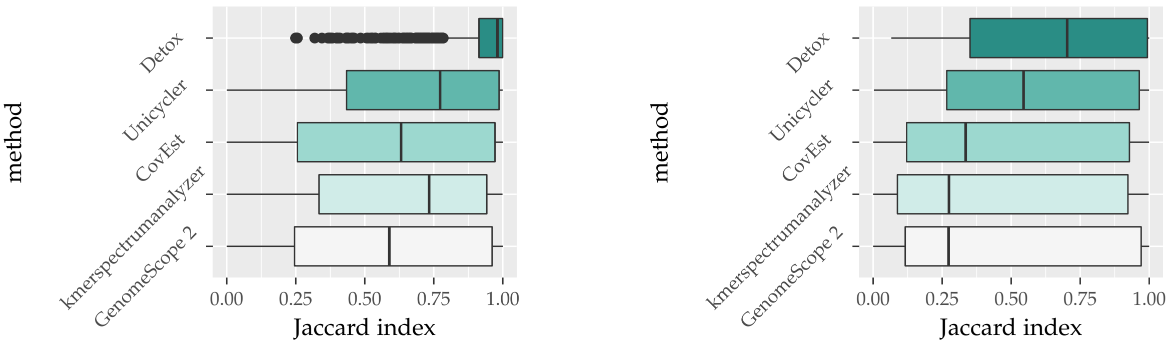
On several 10× sequencing depth datasets, kmerspectrumanalyzer did not provide a model fit, in these cases the Jaccard index was set to 0.

Overall (Fig. 6) we see that Detox performs much better than the other methods. The Jaccard index averages are spread out for all methods, this is because the total number of k-mers in the multiplicity bins for $m>2$ is small and a single missed k-mer or erroneously added k-mer in the estimated bins thus has a larger influence on the Jaccard index. Suppl. Fig. S14 shows the Jaccard index per bin (still averaged over all datasets): Detox outperforms all methods in all multiplicity bins, although we see a lower Jaccard index in all methods in the higher multiplicity bins. All methods perform slightly worse at low sequencing depths similar to the results in Section 3.3. However, the gain obtained by Detox over the other methods is higher at low sequencing depth. As expected, Unicycler performs better than the other k-mer spectrum based methods because it attempts to use some contextual information, albeit in a very localised heuristic manner. Unicycler improves the classification of k-mers with multiplicity 0, 1 and 2 (see Suppl. Fig. S14), but Detox performs even better.

4 Message passing schemes

The order in which messages are passed can greatly influence the rate of convergence of loopy belief propagation. Here we compare several different message passing schemes (see Supplementary Material for pseudo-code) to determine which one provides the best convergence and speed guarantees for our CRFs.

Message passing schemes can be divided into two big categories. Messages can either be passed with synchronous



(a) (b)

Fig. 6. Average Jaccard index (over 6 multiplicity bins) for different multiplicity estimation methods averaged over 180 datasets (5 simulation/subsampling runs for 6 different bacterial organisms for 6 different sequencing depths) for both simulated data (left) and real data (right).

(parallel) updates or with asynchronous (sequential) updates. In the first case there is one pass over all nodes in the factor graph to compute new messages based on the previous iteration and a second pass to update all stored messages. In the second case, each message is updated as soon as it is computed. One message computation according to (4) can thus be based on some messages from the previous iteration and some messages that were already updated during the current iteration. The two most basic asynchronous update schemes are (1) following a fixed order looping over all edges in the factor graph or (2) following a randomised scheme such that the order of updates changes at each iteration.

Sequential updates tend to converge much faster than parallel updates because information is propagated further in the factor graph within fewer iterations. To ensure that computations that propagate more information are performed first, Elidan et al. [28] introduced an informed order of asynchronous message updates called 'maximum residual updates' (MRU). MRU selects which message to send based how much it changes compared to its previous value. This amount of change is estimated based on message residuals:

$$\begin{aligned} &r(m_{I\to i}) \\ &= \left\| \sum_{x_I \in Nb(I)\setminus i} \varphi_I(Nb(I)) \prod_{j\in Nb(I)\setminus i} \prod_{J\in Nb(j)\setminus I} m_{J\to j} \; - \; m_{I\to i} \right\|, \end{aligned}$$

under some norm $\|\cdot\|$. Although there are no theoretical guarantees for the convergence of this scheme, Elidan et al. [28] empirically showed the MRU scheme to converge far more often than parallel or random order sequential updates.

While the maximum residual update scheme has much better convergence properties in terms of number of iterations, the cost of one iteration is significantly larger than for random sequential or parallel belief updates. Sutton et al. [29] showed that the calculations of messages that are only used to update residuals without actually updating messages significantly increase the cost of one message update step. Sutton et al. [29] propose to guide scheduling with an upper bound on the message residuals that is more easily computed, thus significantly reducing the compute time of the maximum residual scheduling, without much influence on the final reached accuracy and convergence properties. The resulting algorithm is called 'Maximum Residual Updates with lookahead 0' (MRU0l). Whereas MRU computes an initial pass over all messages to determine residual initialisation, [29] also provide a different initialisation to the residuals:

$$r(m_{I\to i}) \leq \|\varphi_I(I) - Uniform(I)\|_\infty. \tag{6}$$

This initialisation is derived as the upper bound to the residual of messages initialised to a uniform distribution and updated with the actual factors from the factor graph. We will consider both initialisations in combination with MRU0l.

One frequently observed non-convergent behaviour is oscillating messages, i.e., a message seems to have multiple steady states and with each update this message gets pulled towards the other state, thus causing oscillations. To avoid such behaviour it is possible to use dampened belief updates [30]:

$$m^{(t)}_{I\to i} = (1-\mu)\hat{m}^{(t)}_{I\to i} + \mu m^{(t-1)}_{I\to i}, \qquad 0 \leq \mu \leq 1, \tag{7}$$

with $\hat{m}^{(t)}_{I\to i}$ calculated as in formula (3). Using such dampened updates might result in a successfully converged LBP, whereas normal updates did not. However, in such cases the resulting belief-approximations are often much further away from the true marginals than in the cases where LBP converges without needing dampening.

When oscillating behaviour occurs, this causes only a handful of beliefs to have the largest residual and stay on top of the priority queue. Because of this, messages that have not converged yet, but are lower in the priority queue might never be sent again. Knoll et al. [31] developed two slight alterations to the update schemes to avoid oscillatory behaviour by ensuring that messages that have already been sent often, get a lower priority. We only consider their *weight decay belief propagation* scheme here. This scheme dampens the residuals based on how many times a message has already been selected and sent.

The parallel update, sequential update with random or fixed order and MRU message passing scheme were already implemented in libDAI. Additionally, we imple-

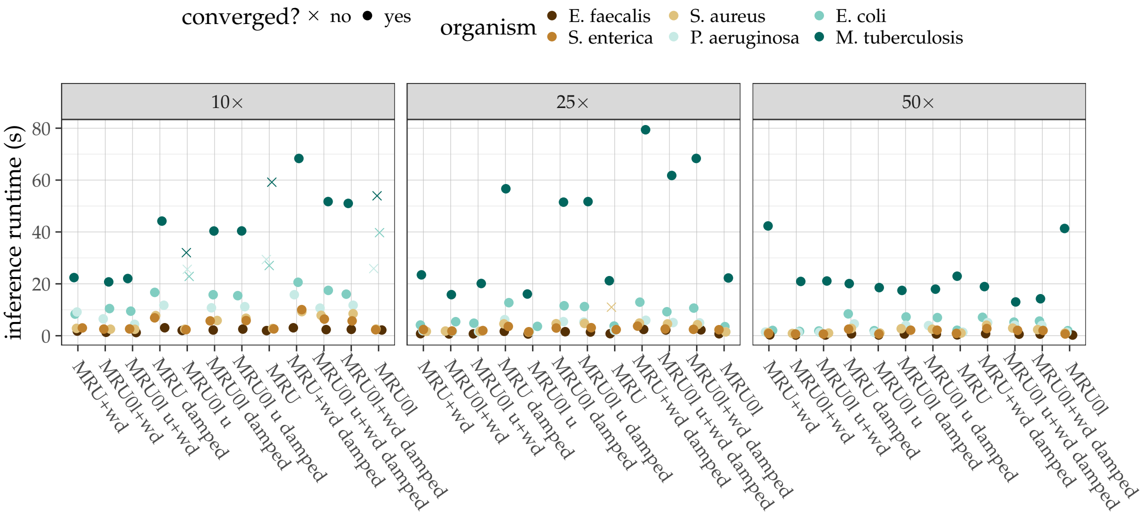


Fig. 7. Total runtime of the inference computations for different message passing schemes. Each observation is the average over 5 different simulations with the same parameter settings. X's are used whenever one of the five runs did not converge (in which case the run used 500 iterations). Results for simulated data at $10\times$, $25\times$ and $50\times$ coverage. MRU: *Maximum Residual Updates*, MRU0l: *Maximum Residual Updates with lookahead 0*, u: uniform distribution-based initialisation of residuals and wd: weight decay.

mented MRU with lookahead 0 and the possibility to add weight decay into the libDAI framework.

4.1 Results and Analysis

We use the simulated datasets from Section 3.3 to compare the convergence properties of different message passing schemes (see Fig. 7). The coverage model is trained using exact inference (subgraphs, $s = 5$). We then use approximate inference with different message passing schemes to determine multiplicity assignments and compare their speed and convergence. Preliminary tests showed that the different MRU-based schemes clearly have better convergence properties than sequential updates with random ordering schemes (results not shown), so we will not show the results for random sequential updates here. Whenever we use dampened updates the dampening factor μ in (7) was set to 0.2. The convergence parameter ε from (5) was set to 10^{-3} in all cases. A stronger convergence criterion was investigated (results not shown), but this had no influence on the final reached accuracy. A maximum of 500 iterations or a maximum runtime of 1800 seconds was allowed before terminating computations.

When they converge, MRU(0l) schemes without weight decay are slightly faster. However, there are several low read support datasets where the MRU(0l) schemes fail to converge within the allotted time. From their final states it is clear that message passing got stuck in oscillating behaviour. Weight decay leads to successful convergence in these datasets. Dampened update schemes also converge, but many more iterations are needed, leading to much larger overall runtimes. While both dampened updates and weight decay scheduling on their own aid in convergence, we do observe that a weight decay scheme combined with dampened updates often leads to many more needed iterations before convergence. On higher read support datasets, the advantage of MRU0l over MRU becomes clear: MRU often needs slightly less iterations than MRU0l, but as the CRFs contain more variables, the faster speed per iteration of MRU0l becomes more advantageous (see Suppl. Fig. S6). Finally, we see that using the uniform distribution upper bound (6) or an initial message computation pass lead to different convergence behaviour. However, there is no clear advantage of one over the other initialisation as one is faster on certain datasets, while the other is faster on other datasets. All message passing schemes showed the same multiplicity accuracy (results not shown). The only time large drops in accuracy are seen is when an approximate inference algorithm did not converge.

We see that the convergence rate improves with higher sequencing depth. At low sequencing depth there is a larger overlap between the distributions that represent the coverage - multiplicity relationship, making several different multiplicities almost equally probable. This can cause regions in the factor graph where convergence is harder. Difficult convergence is often seen in variables that are on a boundary between a region where expected and observed coverage are concordant and a region with many neighbouring nodes and arcs that show a large discrepancy between the expected coverage and the actual observed coverage.

Based on the results observed here, we decide to use MRU0l with weight decay as the default message passing scheme in the Detox pipeline.

5 Discussion

On both real and simulated datasets, we showed that using a single CRF for the entire de Bruijn graph leads to more accurately assigned multiplicities, within a feasible runtime. We observed an increase in accuracy compared to a CRF model for subgraphs, and we outperformed several methods for multiplicity estimation. Additionally, we provided an overview of the most commonly used message passing schemes for Loopy Belief Propagation and empirically showed which ones had good convergence properties for de Bruijn graph based CRFs. Detox was developed for the analysis of short Illumina sequencing reads, which are still most frequently used to analyse smaller genomes. For this reason, the results presented here only analyse bacterial datasets. In Supplementary Material we provide some results on simulated data for larger genomes. These results show the potential and limitations for our method to be used on larger de Bruijn graphs as well, such as those that might arise from metagenomics datasets or long (HiFi) read based de Bruijn graphs. In the remainder of this section, we discuss the use of approximate inference in Detox, in Supplementary Material we discuss possible extensions or adaptations to the inference method.

5.1 When to use approximate inference in the Detox pipeline

The Detox pipeline makes use of the CRF model in three different places (see also Suppl. Fig. 1). In Stage 2 we train the model on a subset of the nodes and arcs. Approximate inference computes a single CRF for the entire de Bruijn graph, thus computing all probabilities for all nodes and arcs and having no benefit from only retrieving a subset of the probabilities for training. We observe only a small improvement in accuracy when using a model trained on all nodes and arcs with approximate inference, leading us to believe that the parameter estimation is mostly guided by easy to determine multiplicities that are characterised well by exact inference on a subgraph. Additionally, preliminary tests on the use of approximate inference in this stage showed that convergence issues could arise more easily in the first EM-step due to a less than optimal initialisation. Therefore, we choose not to implement approximate inference in Stage 2. During Stage 3 graph cleaning we also determine multiplicities for a subset of the variables, i.e., those nodes and arcs that have low coverage. We noticed more convergence issues on low coverage datasets when sequencing errors are still present. We choose to use subgraphs of a reasonable size to determine the multiplicity estimates on which node and arc removal is based. In the final pass of Stage 3 it is most useful to use approximate inference. Due to all the previous cleaning steps, a smaller number of nodes and arcs remains (see Suppl.Table S1), and we observe very good convergence properties of the approximate inference algorithms. Moreover, the remaining high multiplicity nodes are often more connected with each other and more connections between nodes result in an exponential increase in runtime by a higher factor. The potential speedup by replacing exact by approximate inference is thus much larger. Additionally, high multiplicity nodes are harder to characterise correctly with inference on subgraphs. We, therefore, believe the use of approximate inference is most valuable in the final step of Stage 3 to compute all multiplicities of remaining nodes and arcs.

5.2 The impact of non-convergence on the accuracy of multiplicity assignments

Note how the convergence criterion (5) is formulated as a maximum over the change in belief of all variables. Non-convergence almost always means that the beliefs of a small proportion of variables keep oscillating causing the convergence criterion not to be fulfilled. However, the bulk of the variable beliefs have converged already at this point. Especially using the weight decay criterion, all parts of the factor graph are successfully explored before the stopping criterion is reached. Even in cases of non-convergence a good accuracy can thus be reached. This is observed in our results as well. In Section 3.3 approximate inference in the Detox pipeline converged on 355 of 360 datasets. In datasets with low sequencing depth, because of coverage gaps, the de Bruijn graph can contain one main connected component representing the bulk of the genome and additional small connected components, disconnected from the main graph. In these cases we build a CRF model for each connected component. In 4 out of the 5 cases, non-convergence occurred in a CRF model for such a small component, 500 iterations were reached, but runtime remained short. The CRF model for the main bulk of the de Bruijn graph did converge successfully. As a result, only the beliefs for < 10 variables did not converge, while all other beliefs did converge and the total influence on the accuracy is thus minimal. In the fifth case, non-convergence did occur in the CRF model for the main component of the de Bruijn graph, but here again this meant only the beliefs of around 20 variables kept oscillating until 500 iterations were reached (total runtime remained less than 1 minute), while all other beliefs had converged. In all 5 non-convergence cases we, therefore, saw no significant difference in accuracy in comparison with the other 4 runs on datasets with the same parameters.

6 Conclusion

We presented a comparison of different Loopy Belief Propagation message passing schemes for inference on CRFs with higher order interactions that model a de Bruijn graph. From this empirical evaluation we could conclude that the Maximum Residual Update scheme with lookahead 0 supplemented by weight decay ordering converges successfully on CRF models for de Bruijn graphs of a broad range of complexity. Besides guiding our own choice of message passing scheme, we hope this overview can highlight the advantages and disadvantages of using certain schemes on CRFs with higher order interactions. We then used the best performing message passing scheme to implement approximate inference in the Detox pipeline such that we can use a single CRF model to represent the entire de Bruijn graph. Multiplicity estimates based on this model were more accurate than the ones using inference on subgraphs. And, by using approximate inference techniques our computations were performed within reasonable time. The resulting de Bruijn graph is an accurate representation of the genome and can be used for accurate repeat resolution to obtain a high quality assembly of the genome.

Acknowledgments

AS is funded by a Ph.D. grant of the Flanders Research Foundation (File number: 1174621N).

References

[1] P. A. Pevzner, H. X. Tang, and M. S. Waterman, "An Eulerian path approach to DNA fragment assembly," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 98, no. 17, pp. 9748–9753, aug 2001.

[2] P. E. C. Compeau, P. A. Pevzner, and G. Tesler, "How to apply de Bruijn graphs to genome assembly," *Nature Biotechnology*, vol. 29, no. 11, pp. 987–991, nov 2011.

[3] D. R. Zerbino and E. Birney, "Velvet: Algorithms for de novo short read assembly using de Bruijn graphs," *Genome Research*, vol. 18, no. 5, pp. 821–829, may 2008.

[4] X. Li and M. S. Waterman, "Estimating the repeat structure and length of DNA sequences using l-tuples," *Genome research*, vol. 13, no. 8, pp. 1916–1922, 2003.

[5] D. Williams, W. L. Trimble, M. Shilts, F. Meyer, and H. Ochman, "Rapid quantification of sequence repeats to resolve the size, structure and contents of bacterial genomes," *BMC Genomics*, vol. 14, aug 2013.

[6] M. Hozza, T. Vinar, and B. Brejova, "How Big is that Genome? Estimating Genome Size and Coverage from k-mer Abundance Spectra," in *International Symposium for String Processing and Information Retrieval*, ser. Lecture Notes in Computer Science, vol. 9309. Springer, 2015, Proceedings Paper, pp. 199–209.

[7] T. R. Ranallo-Benavidez, K. S. Jaron, and M. C. Schatz, "GenomeScope 2.0 and Smudgeplot for reference-free profiling of polyploid genomes," *Nature communications*, vol. 11, no. 1, pp. 1–10, 2020.

[8] S. Sarmashghi, M. Balaban, E. Rachtman, B. Touri, S. Mirarab, and V. Bafna, "Estimating repeat spectra and genome length from low-coverage genome skims with RESPECT," *PLoS computational biology*, vol. 17, no. 11, p. e1009449, 2021.

[9] A. Steyaert, P. Audenaert, and J. Fostier, "Accurate determination of node and arc multiplicities in de bruijn graphs using conditional random fields," *BMC Bioinformatics*, vol. 21, no. 1, p. 402, 2020.

[10] M. Heydari, G. Miclotte, P. Demeester, Y. Van de Peer, and J. Fostier, "Evaluation of the impact of Illumina error correction tools on de novo genome assembly," *BMC Bioinformatics*, vol. 18, aug 2017.

[11] J. T. Simpson and R. Durbin, "Efficient de novo assembly of large genomes using compressed data structures," *Genome Research*, vol. 22, no. 3, pp. 549–556, mar 2012.

[12] I. MacCallum, D. Przybylski, S. Gnerre, J. Burton, I. Shlyakhter, A. Gnirke, J. Malek, K. McKernan, S. Ranade, T. P. Shea, L. Williams, S. Young, C. Nusbaum, and D. B. Jaffe, "ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads," *Genome Biology*, vol. 10, no. 10, 2009.

[13] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, A. V. Pyshkin, A. V. Sirotkin, N. Vyahhi, G. Tesler, M. A. Alekseyev, and P. A. Pevzner, "SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing," *Journal of Computational Biology*, vol. 19, no. 5, pp. 455–477, may 2012.

[14] A. Bankevich, A. V. Bzikadze, M. Kolmogorov, D. Antipov, and P. A. Pevzner, "Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads," *Nature biotechnology*, pp. 1–7, 2022.

[15] P. A. Pevzner and H. Tang, "Fragment assembly with double-barreled data," *Bioinformatics*, vol. 17, no. suppl 1, pp. S225–S233, 2001.

[16] R. R. Wick, L. M. Judd, C. L. Gorrie, and K. E. Holt, "Unicycler: Resolving bacterial genome assemblies from short and long sequencing reads," *PLOS Computational Biology*, vol. 13, no. 6, pp. 1–22, 2017.

[17] Y. Suzuki and G. Myers, "Accurate k-mer Classification Using Read Profiles," in *22nd International Workshop on Algorithms in Bioinformatics (WABI 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[18] M. J. Wainwright, M. I. Jordan, and Others, "Graphical models, exponential families, and variational inference," *Foundations and Trends in Machine Learning*, vol. 1, no. 1–2, pp. 1–305, 2008.

[19] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. Massachusetts Institute of Technology, 2009.

[20] B. Yu and Z. Fan, "A comprehensive review of conditional random fields: variants, hybrids and applications," *Artificial Intelligence Review*, vol. 53, no. 6, pp. 4289–4333, 2020.

[21] M. C. Cariello, A. Lenci, and R. Mitkov, "A comparison between named entity recognition models in the biomedical domain," in *Proceedings of the Translation and Interpreting Technology Online Conference*, 2021, pp. 76–84.

[22] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, feb 2001.

[23] J. M. Mooij, "libDAI: A Free and Open Source C++ Library for Discrete Approximate Inference in Graphical Models," *Journal of Machine Learning Research*, vol. 11, pp. 2169–2173, 2010.

[24] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Constructing free-energy approximations and generalized belief propagation algorithms," *IEEE Transactions on Information Theory*, vol. 51, no. 7, pp. 2282–2312, jul 2005.

[25] W. Huang, L. Li, J. R. Myers, and G. T. Marth, "ART: a next-generation sequencing read simulator," *Bioinformatics*, vol. 28, no. 4, pp. 593–594, feb 2012.

[26] R. Chikhi, A. Limasset, and P. Medvedev, "Compacting de Bruijn graphs from sequencing data quickly and in low memory," *Bioinformatics*, vol. 32, no. 12, pp. 201–208, jun 2016.

[27] M. Kokot, M. Długosz, and S. Deorowicz, "KMC 3: counting and manipulating k-mer statistics," *Bioinformatics*, vol. 33, no. 17, pp. 2759–2761, 2017.

[28] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: Informed scheduling for asynchronous message passing," *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pp. 165–173, 2006.

[29] C. Sutton and A. McCallum, "Improved dynamic schedules for belief propagation," *arXiv preprint arXiv:1206.5291*, 2012.

[30] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *uncertainty in artificial intelligence, proceedings*, 1999, pp. 467–475.

[31] C. Knoll, M. Rath, S. Tschiatschek, and F. Pernkopf, "Message Scheduling Methods for Belief Propagation," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Artificial Intelligence, vol. 9285. Springer, 2015, Proceedings Paper, pp. 295–310.

Aranka Steyaert Aranka Steyaert received her M.Sc. degree in Mathematics from Ghent University in 2017. Currently, she is working as a Ph.D. researcher at the faculty of Engineering and Architecture in the IDLab research group, Ghent University - imec. Her research focuses on Probabilistic Graphical Models and the development of bioinformatics algorithms.

Pieter Audenaert Pieter Audenaert received his M.Sc. and Ph.D. in pure mathematics from Ghent University, in 2000 and 2004 respectively. Currently, he is a professor at Ghent University - imec and works in the field of networks in its broadest sense: from communication networks and logistic networks, to protein-interactions and social networks. To this aim, he specializes in graph theory and mathematical algorithms with a focus on applying theoretical results in the field of computer science. This entails e.g. datamodeling, computational analysis and statistical forecasting.

Jan Fostier received the M.Sc. degree in Engineering Physics in 2005 and the Ph.D. degree in Applied Physics in 2009, both from Ghent University, Belgium. Currently, he is associate professor at the IDLab research group at Ghent University - imec. His research interests are fast algorithms, numerical techniques and distributed computing for bioinformatics problems.