Repeat-aware Comparative Genome Assembly

Peter Husemann* and Jens Stoye
AG Genominformatik, Technische Fakultät, Bielefeld University, Germany

{phuseman, stoye}@techfak.uni-bielefeld.de

Abstract: The current high-throughput sequencing technologies produce gigabytes of data even when prokaryotic genomes are processed. In a subsequent assembly phase, the generated overlapping reads are merged, ideally into one contiguous sequence. Often, however, the assembly results in a set of contigs which need to be stitched together with additional lab work. One of the reasons why the assembly produces several distinct contigs are repetitive elements in the newly sequenced genome. While knowing order and orientation of a set of non-repetitive contigs helps to close the gaps between them, special care has to be taken for repetitive contigs. Here we propose an algorithm that orders a set of contigs with respect to a related reference genome while treating the repetitive contigs in an appropriate way.

1 Introduction

The sequencing of genomes has become easier and cheaper with the current massively parallel sequencing methods [Mar08]. Following a shotgun approach, these methods fragment the genome randomly into small parts. The ends of those fragments are sequenced and referred to as *reads*, both reads of one fragment form a *mate pair*. In a subsequent assembly phase, an assembler software tries to merge overlapping parts of the reads into longer contiguous sequences [Pop09], the so called *contigs*. If the size of the fragments is known, the mate pairs have a defined distance and this information can also help in the assembly. Ideally, the result is a single sequence which resembles the complete genome. However, there are a few obstacles that lead to several contigs instead of a single one. Besides non-random fragmentation and areas of an unusual GC content, repeats play a major role in this process [MKS10]. For the latter the assembly software can not distinguish between the reads from different occurrences of the repeating region. Thus, reads with the same sequence from several distinct origins are merged together to a single *repetitive contig*.

In general, the output of the assembly phase is a set of contigs and maybe a rough scaffold derived for example from mate pair information. In order to retrieve the complete genomic sequence, the gaps between the contigs have to be filled by running additional experiments in the lab. If two contigs are known to be adjacent, then it is possible to close the gap with a (long range) PCR or with primer walking for even larger gaps. For the tedious process of gap closing it is therefore beneficial to know the *layout* of the contigs, meaning the order

and orientation of them. Mate pair information can again help in this situation, but it fails for gaps longer than the fragment size, and it is also not reliable on repetitive sequences.

Fortunately, many genomes have already been sequenced completely and if one of them is related to the newly sequenced genome then it can be used as a reference to estimate a proper layout. There are a few programs that deal with this task: Projector2 [vHZKK05] maps the contigs on a single template genome and designs primer pairs for gap closure, OSLay [RSH07] finds an optimal syntenic layout, other programs like PGA [ZZLB08] and treecat [HS10a] are even able to utilize several reference genomes to predict a consensus layout. Still problematic for the above mentioned programs are major rearrangements in the reference genomes as well as repeating regions. To cope with the latter, all programs except for treecat employ or at least suggest a repeat masking step.

In this paper we address the problem of repeating contigs in prokaryotic genomes with the goal to find a better layout for a set of contigs according to a closely related reference genome. Therefore we present a novel algorithm that includes repeating contigs as often as necessary in a computed layout. This greatly reduces the complexity of the layout graph while not removing the repetitive contigs.

After introducing the basic concept of a contig adjacency graph in Section 2, we address in Section 3 how repeating contigs can be discovered from a given set of contigs. In Section 4 we present a specially designed algorithm that uses the repeat information to estimate a more appropriate layout for the contigs. Section 5 contains an evaluation of the algorithm and a comparison of the repeat integration with other contig ordering programs.

2 Contig adjacency graph

Let $\Sigma = \{A, C, G, T\}$ be the alphabet of nucleotides. We denote by Σ^* the set of all finite strings over Σ, by $|s| := \ell$ the *length* of string $s = s_1 \dots s_\ell$, and by $s[i, j] := s_i \dots s_j$ with $1 \leq i \leq j \leq \ell$ the *substring* of s that starts at position i and ends at position j.

Suppose we are given a set of contigs $\mathcal{C} = \{c_1, \dots, c_n\}$, $c_i \in \Sigma^*$, and a reference genome $g \in \Sigma^*$ that has already been finished. The *contig adjacency graph* for these sequences is then the weighted graph $G_{\mathcal{C},g} = (V, E)$ that contains for each contig $c_i \in \mathcal{C}$ two vertices: l_i as the *left connector* and r_i as the *right connector* of contig c_i, thus $V = \{l_1, \dots, l_n, r_1, \dots, r_n\}$. A function $contig(v)$ refers to the contig for which vertex v represents the left or right connector. The graph $G_{\mathcal{C},g}$ is fully connected, $E = \binom{V}{2}$, and we term $A = \{\{v, v'\} \mid contig(v) \neq contig(v')\}$ the set of *adjacency edges* that connect the contigs among each other.

The edge weights are given by a function $w : E \to \mathbb{R}_0^+$, and we are particularly interested in the weights of the adjacency edges A. These shall provide a score of how likely the involved connectors are adjacent with respect to the reference genome. One method to calculate the weights based on matching the contigs onto the reference genome is described in [HS10a]. There, the pairwise matches from different contigs are used to calculate a score for the adjacency of the involved contig connectors. While the scores increase with the length of the corresponding matches, they are weighted by their distance. A high

weight $w(\{r_i, l_j\})$, for example, supports that the right connector (or head) of contig c_i is adjacent to the left connector (or tail) of contig c_j. We call the sum of all edge weights of a particular node $v \in V$ the *total support* of that node, denoted by $\mathcal{S}_v = \sum_{v' \in V} w(\{v, v'\})$. To estimate how significant an adjacency edge $e = \{v, v'\} \in A$ is for a given contig connector $v \in V$, we consider the *relative support*: $\mathcal{S}_v^{\mathsf{rel}}(e) = \frac{w(e)}{\mathcal{S}_v}$. Intuitively, this fraction tells how specific the connection is for the given contig connector. A single high weight edge results in a relative support close to one, while many equally good connections will lower the value. Note that in general $\mathcal{S}_v^{\mathsf{rel}}(\{v, v'\}) \neq \mathcal{S}_{v'}^{\mathsf{rel}}(\{v, v'\})$.

Given a contig adjacency graph, a natural task is to find a subgraph of it that contains all relevant adjacencies in order to ease the gap closure phase of the sequencing project. We call any subgraph with this property a *layout graph* of a set of contigs. A basic approach could be to find a tour of maximal weight that contains each contig once and in a specified direction. This leads essentially to the problem of finding a longest Hamiltonian cycle in $G_{\mathcal{C},g}$ and is thus NP hard. Moreover, a meaningful biological result can differ from it, especially if some contigs appear several times on the genome. In this case, a repetitive contig has to be included several times into an adequate layout. In the next section we describe how to detect such repetitive contigs.

3 Repeat detection

Our approach for contig ordering distinguishes between repetitive and non-repetitive contigs. Assuming that repeating regions are conserved between closely related species, we can determine if a contig $c \in \mathcal{C}$ is repetitive by matching it onto a given reference genome g. A *match* of c in g is represented as a pair $m = \big((s_b, s_e), (t_b, t_e)\big)$ where the indices denote the starting and ending positions of the two substrings $c[s_b, s_e]$ and $g[t_b, t_e]$. An alignment of the substrings is supposed to yield a high score. Reverse complement matches can be modeled with this notation as well, but are left out for simplicity.

Given a set of matches $\mathcal{M}_{c,g}$ of contig c on the reference genome g, we can determine which matches are repetitive and from this derive if the contig occurs repetitively: We call a match $m = \big((s_b, s_e), (t_b, t_e)\big) \in \mathcal{M}_{c,g}$ *repetitive* if there exists another match $m' = \big((s'_b, s'_e), (t'_b, t'_e)\big) \in \mathcal{M}_{c,g}$ such that (i) the contig substring of m is included in the substring of m' ($s_b \geq s'_b$ and $s_e \leq s'_e$), and (ii) the match positions on the reference are not overlapping ($\{t_b, \ldots, t_e\} \cap \{t'_b, \ldots, t'_e\} = \emptyset$). The exact positions of the matches may vary for different matching procedures and/or scoring functions, so we allow for condition (i) a slack of ρ_1 times the length of m. By default we use a value of 10% for ρ_1.

We call a contig *repetitive* if it has at least one repetitive match m of sufficient length. Sufficient means that at least a fraction of ρ_2 of the contig is covered by the repetitive match: $s_e - s_b \geq \rho_2 \cdot |c|$. As default we set $\rho_2 = 0.9$. We call all contigs that are not repetitive for the sake of a shorter notation *regular contigs*.

Contigs that are repetitive on the newly sequenced genome are not necessarily repetitive on the employed reference genome. To extend, as well as verify, the prediction of repetitive contigs, one can use the information on how many reads have been merged to form a

contig provided from the assembly phase. For each contig the average read coverage can be calculated, and by looking at the deviation from the median of these values, it can be observed if a contig is over- or underrepresented with reads. Highly overrepresented contigs are most likely repetitive since the reads gathered from all repeat occurrences are merged to a single contig. Even more, the ratio with respect to the median can serve as a rough estimate for the number of occurrences of a repetitive contig.

4 Repeat-aware layout algorithm

In this section we adopt the contig adjacency discovery algorithm that was proposed in [HS10a] to be aware of repetitive contigs and include them appropriately. The overall strategy is to distinguish between regular and repetitive contigs and to process both sets one after another. The absence of repetitive contigs in the first set implies that most contigs should have exactly two neighbors. Following this observation, we describe in Section 4.1 an algorithm for creating a basic layout graph. In the subsequent Section 4.2 we address how this layout graph can be augmented with the repetitive contigs in a meaningful way.

4.1 Layouting the non-repetitive parts of the genome

To devise a basic layout graph for the regular contigs, two steps are necessary:

1. The contig adjacency graph that contains the edge weights for all contig connectors has to be computed. Note that the graph is created for repetitive *and* regular contigs, thus the procedure starts with matching *all* contigs onto the given reference genome. The score calculation is performed as in [HS10a] with the difference that repetitive matches are ignored for the calculation of scores between regular contigs. This helps to reduce misleading edges for a contig caused for example if it is flanked by repeats. Of course, for repetitive contigs all matches are used.

2. In the second step, the calculated edge weights can be used to extract the adjacencies with the highest support and collect them in a layout graph. We want to discover those adjacencies from the contig adjacency graph that are most likely present in the true order of the regular contigs. Since we do not resolve repetitive contigs at this stage, the result should be a set of linear chains of the contigs which can also be present in the form of one or several cycles. Our algorithm processes all edges between regular contigs in decreasing weight order and greedily integrates them one by one into an initially empty layout graph, except if any of the involved contig connectors is already used. With this heuristic approach we generate multiple fragments of good adjacencies that are in general joined to larger chains during the course of the algorithm.

This procedure finds for most regular contigs appropriate neighbors. However, if very small contigs lie between two large contigs then we sometimes observe a *shadowing effect*,

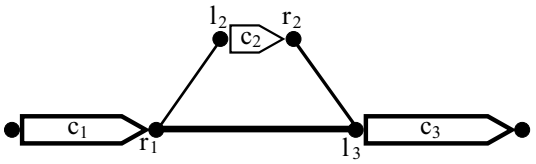

Figure 1: Shadowing effect: If a small contig c_2 is on a reference genome located between the larger contigs c_1 and c_3, in the contig adjacency graph the correct edges to c_2 can have a lower weight than the edge $\{r_1, l_3\}$.

as illustrated in Figure 1: The adjacency edge between the large contigs can have a high weight that shadows the edge weights to the small contig. Thus, the algorithm would not include the small contig into the layout graph. This behavior is generally unwanted but, as we will see in Section 4.2, it can be advantageous for small repetitive contigs. That is why we do not abandon the effect, e. g. by ignoring the size of the matches in the weight function. Instead, we compensate the shadowing effect for the affected regular contigs by integrating them into the initial layout as good as possible. Therefore, we look at all edges that were not integrated in step 2. Again, in decreasing weight order we include an edge if any of the two connectors is still unused in the layout. To control that only very specific edges edges are incorporated, we test if the additional edge has a high relative support $\mathcal{S}^{\mathsf{rel}}$ of at least τ_1. Although the shadowing edge stays in the layout, in most cases the correct edges from the small contig will also be included, resulting in a triangle shape of connections as in Figure 1.

4.2 Adding the repetitive contigs

Starting with the basic layout graph of the previous subsection, the task is now to include the repetitive contigs into the layout. For genome finishing, the gain through repetitive contigs is only limited since they are not well suited for a primer-based closing of gaps. Primers for the repetitive sequence will bind unspecifically to several regions on the genome and should thus be avoided. Nonetheless, we believe that it is very helpful in the finishing phase of a sequencing project for a researcher to be informed which repetitive contigs interrupt the gap between two regular contigs. However, the order of the repetitive contigs in a gap plays, to our opinion, only a secondary role because this information can not directly help in the finishing process: If both primers are based on repetitive contigs, this will produce even more unpredictable results. Our idea in Algorithm 1 is therefore to place each repetitive contig as often as necessary between the corresponding regular contigs into the basic layout graph.

The important edges that we want to integrate in our basic layout are those which connect a repetitive contig with a regular one, see line 1. We demand that the relative support of these edges with respect to the repetitive contig is higher than a threshold τ_2. This avoids the incorporation of arbitrarily weak edges. The edges between repetitive contigs are not

Algorithm 1: Repetitive contigs integration algorithm

Input: set of contigs $\mathcal{C}$, set of repetitive contigs $\mathcal{R} \subset \mathcal{C}$, contig adjacency graph $G = (V, E)$, basic layout graph G_L

Output: repeat-aware layout graph G_L of the contigs

1 let $E_{\text{rep}} = \{\{v, v'\} \mid contig(v) \in \mathcal{R}$, $contig(v') \notin \mathcal{R}$ and $\mathcal{S}_v^{\text{rel}}(\{v, v'\}) > \tau_2\}$
2 **foreach** edge $e \in E_{\text{rep}}$, sorted by decreasing weight $w(e)$ **do**
3 **if** $e = \{v_1, l\}$ contains the left connector l of a contig $c \in \mathcal{R}$ **then**
4 let r be the right connector of contig c
5 **if exists** $v_2 = \arg\max_{v \in V} \{w(\{v_1, v\}) \mid \{r, v\} \in E_{\text{rep}}\}$ **then**
6 duplicate l and r to l' and r'
7 $V_L = V_L \cup \{v_1, l', r', v_2\}$
8 $E_L = E_L \cup \{\{v_1, l'\}, \{l', r'\}, \{r', v_2\}\}$
9 remove $\{v_1, l\}$ and $\{r, v_2\}$ from E_{rep}
10 **end**
11 **else** // $e = \{r, v_1\}$ *contains the right connector of a contig* $c \in \mathcal{R}$
12 perform lines 4 to 10 analogously
13 **end**
14 **end**

considered in this approach, as motivated above. For the interesting edges, we try to find for each involved regular contig connector a suitable counterpart that is also connected to the other end of the repetitive contig, as shown in lines 4 to 10 for the left connectors. This procedure is based on the following observation: As illustrated in Figure 2, a repetitive contig $c \in \mathcal{R}$ usually has several good edges for its right and its left connector leading to different regular contigs. The problem is to determine which edges belong to a particular repeat occurrence on the reference genome. The shadow effect, which was an obstacle

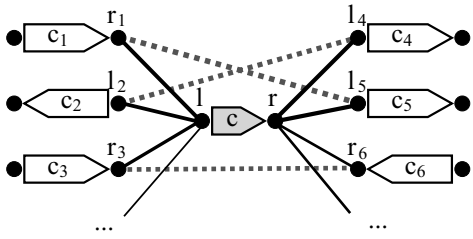

Figure 2: Typical scenario for the adjacency edges of a repetitive contig $c \in \mathcal{R}$. The dashed lines depict the best edge from a contig connector on the right to a contig connector on the left.

for regular contig ordering, becomes here now an advantage. In the example of Figure 2, the edge $\{r_1, l_5\}$ has a high weight if the contigs c_1 and c_5 are only separated by the occurrence of the relatively small repeating contig c. The strategy is hence to search, for any regular node that is connected to one side of a repetitive contig, for a counterpart that

is connected to the other side, such that the edge from the node to the counterpart has the highest weight.

This way, we find for each significant occurrence of a repetitive contig the two surrounding regular contigs with respect to the reference genome. For all occurrences we add two new connectors of the repetitive contig and the appropriate edges to the layout graph.

5 Results

We evaluated our algorithm on two biological datasets. The aim was to correctly place the repetitive contigs between the regular contigs allowing them to appear more than once. None of the programs for comparative contig arrangement that we are aware of was designed to handle repetitive contigs explicitly. Still, we applied some of them to our data in order to see whether they would recover a part of the connections nevertheless. After introducing the datasets, we will explain the evaluation procedure and then show the results.

Dataset For the evaluation we prepared a set of contigs and acquired two reference genomes. All genomic sequences belong to the *Corynebacteria* genus. The contigs originate from a 454 sequencing run of the *Corynebacterium urealyticum* strain DSM7109 conducted at the Center for Biotechnology (CeBiTec) of Bielefeld University. The assembly yielded 223 contigs with a total size of 2 316 966 bases. We discarded all 'contigs' with a size of less than 500 bases resulting in a set of 69 contigs. This step was taken since many small contigs that consist of only two or three reads can be very confusing in the mapping process. Nevertheless, the N50 contig size, which is a more robust characterization for the size distribution of contig sets than the mean or median, stays the same for both sets.
As reference sequences we took the already finished genome of *Corynebacterium urealyticum* strain DSM7109 [TTT$^+$08] (NCBI Number NC_010545) and the closely related genome of *Corynebacterium jeikeium* K411 (NC_007164).

To ease the evaluation we renumbered and renamed the 69 contigs. Therefore, the contigs were mapped onto the perfect reference, *C. urealyticum*, and ordered according to their matches using the tool r2cat [HS10b]. The program revealed that 15 contigs are repetitive while the remaining 54 contigs were regular according to Section 3. The regular contigs were renumbered consecutively in their true order such that adjacencies can easily be seen. The repetitive contigs were numbered in arbitrary order and prefixed with the letter 'r'. As the next step we manually inspected the matches using r2cat and noted for each pair of adjacent regular contigs which repetitive contigs have an occurrence between them.

Experimental Setup The manually annotated list of repeating contigs between regular contigs serves for the experiments as a standard of truth. To see which of these connections can actually be discovered, we applied the following programs on the described datasets: Projector2 [vHZKK05], OSLay [RSH07], PGA [ZZLB08], treecat [HS10a] and our new

Table 1: Experimental results of ordering the repetitive *C. urealyticum* contigs. Each program was applied on the described datasets. The results of PGA were varying, since it is a randomized algorithm, so we give the mean values for applying the program 20 times.

	perfect reference				*C. jeikeium* reference			
Program	TP	FP	TPR	PPV	TP	FP	TPR	PPV
Projector2	10.0	0.0	0.06	1.00	1.0	5.0	0.01	0.17
OSLay	15.0	1.0	0.09	0.94	4.0	2.0	0.03	0.67
PGA	24.4	16.1	0.15	0.60	20.4	27.9	0.13	0.42
treecat	29.0	1.0	0.18	0.97	20.0	8.0	0.13	0.71
repcat	140.0	7.0	0.89	0.95	54.0	37.0	0.34	0.59

algorithm *repcat* (repeat-aware contig arrangement tool) which is derived from treecat. All programs were used with their standard parameters as proposed in the publication or as pre-given on the web-service unless otherwise stated. At the webform of Projector2 we switched off repeat masking for contigs and target genome and reduced the minimum contig size to 500 bases such that all contigs could be considered. PGA, unfortunately, filters all contigs smaller than 3.5 kb and discards this way all repetitive contigs, so we modified their perl script to include all contigs. The algorithm from *repcat* is adopted from treecat and inherits some of its parameters for the contig adjacency graph creation: We set the standard deviation of the insertion/deletion size to $\sigma_1 = 2\,000$ bases and the lost fragment weighting factor φ to 0. Furthermore, for the repeat detection we used the default parameters stated in Section 3 and selected for the layouting the experimentally evaluated parameters $\tau_1 = 90\%$ and $\tau_2 = 0.1\%$.

In the following evaluation, we assess how well repetitive contigs can be integrated into an ordering. As motivated in Section 4.2, the interesting connections are those from a repetitive contig to a regular contig. Therefore, we extracted those connections from the output of the programs and compared them with our manually annotated standard of truth. If a repetitive contig is present in between a gap, we count the connections to the corresponding regular contigs as true positives (TP). If an adjacency is not given in our list, we count this as a false positive (FP). For example, if the repetitive contigs r008, r012, and r013 are between 048 and 049, we count a connection from r013 to 048 as TP, whereas a connection from r003 to 049 is a FP, except if for example r003 would occur between 049 and 050 as well.

Evaluation We ran all programs on both datasets and counted the TP and FP values as described above. The manually annotated repeat list revealed that the 15 repetitive contigs occurred in 79 instances on the genome. For each occurrence two true positive connections could be predicted, so the sum of all positive predictions is $P = 158$. Given TP, FP and P we calculated the *sensitivity* (also called *true positive rate*, $TPR = \frac{TP}{P}$) and the *precision* (also called *positive predictive value*, $PPV = \frac{TP}{TP+FP}$) of the predicted connections. Table 1 shows the resulting values for each program and dataset.

We would like to note here that a direct comparison of the values between the different programs has to be handled with caution. While our algorithm was specially built to include repeating contigs as often as necessary into a layout, the objectives of Projector2 and OSLay are to devise a linear ordering where each contig occurs exactly once. These programs can generate at most two true positive connections per repetitive contig, that is a maximum of $\text{TP}_{max} = 30$ for our dataset. PGA combines five such linear layouts and achieves thus at most $\text{TP}_{max} = 150$ correct connections. The program treecat is not restricted by this number but will stop to add edges if both connectors of a repetitive contig have been integrated into a layout. Our new algorithm can in principle predict two true positive connections for *every occurrence* of a repetitive contig, thus gaining a clear advantage over the other programs in this setting. Regardless of that, we believe that an appropriate placing of repetitive contigs is very helpful for a sequencing project and the alternative to mask and discard repeats is not a sufficient solution.

The results for the perfect reference show that the predictions are in general quite accurate. Projector2 and OSLay find only a fraction of their TP_{max}. PGA and treecat recover some more true positives, but PGA surprises with a rather high number of false positives. Our new algorithm *repcat* recovers nearly 90% of the possible true positive connections. This is somehow expected, since it is the only of the applied methods that handles repeats explicitly.

For the more realistic reference *C. jeikeium*, the true positives decrease for all programs while the false positives increase. Especially the results of our algorithm are hit by this tendency, although it still finds many more of the correct repeat adjacencies than any of the other programs.

6 Conclusion

In the context of ordering contigs to assist the gap closure of prokaryotic sequencing projects, we propose a novel algorithm that includes an explicit handling of repetitive contigs. While the common objective of related applications is to find a linear layout, this is obviously not feasible for repetitive contigs. Hence, our approach orders the non-repetitive contigs first and then integrates all repetitive contigs in between the gaps, as often as necessary. We believe that this strategy is more adequate than discarding all repetitive contigs since it allows to assess which of these sequences should be expected in the gaps.

In this setting, the contig adjacency graph turns out to be a valuable concept that is flexible enough to be extended to handle repetitive contigs. However, the reduction to a layout graph always contains the risk of losing important adjacencies. An interactive visualization of the whole graph could help to unleash its true potential.

Considering the problem of repetitive contigs, a next step could be to verify to which degree of synteny an ordering of repetitive elements is feasible and if the information from several reference genomes helps or maybe even confuses.

Acknowledgments

We thank Eva Trost and Andreas Tauch for providing the sequence data.

References

[HS10a] P. Husemann and J. Stoye. Phylogenetic comparative assembly. *Algorithms Mol. Biol.*, 5(1):3, 2010.

[HS10b] P. Husemann and J. Stoye. r2cat: synteny plots and comparative assembly. *Bioinformatics*, 26(4):570–571, 2010.

[Mar08] E. R. Mardis. The impact of next-generation sequencing technology on genetics. *Trends Genet.*, 24(3):133–141, 2008.

[MKS10] J. R. Miller, S. Koren, and G. Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, 2010.

[Pop09] M. Pop. Genome assembly reborn: recent computational challenges. *Brief. Bioinform.*, 10(4):354–366, 2009.

[RSH07] D. C. Richter, S. C. Schuster, and D. H. Huson. OSLay: optimal syntenic layout of unfinished assemblies. *Bioinformatics*, 23(13):1573–1579, 2007.

[TTT+08] A. Tauch, E. Trost, A. Tilker, U. Ludewig, S. Schneiker, A. Goesmann, W. Arnold, T. Bekel, K. Brinkrolf, I. Brune, S. Götker, J. Kalinowski, P.-B. Kamp, F. Pereira Lobo, P. Viehoever, B. Weisshaar, F. Soriano, M. Dröge, and A. Pühler. The lifestyle of *Corynebacterium urealyticum* derived from its complete genome sequence established by pyrosequencing. *J. Biotechnol.*, 136(1-2):11–21, 2008.

[vHZKK05] S. A. F. T. van Hijum, A. L. Zomer, O. P. Kuipers, and J. Kok. Projector 2: contig mapping for efficient gap-closure of prokaryotic genome sequence assemblies. *Nucleic Acids Res.*, 33:W560–W566, 2005.

[ZZLB08] F. Zhao, F. Zhao, T. Li, and D. A. Bryant. A new pheromone trail-based genetic algorithm for comparative genome assembly. *Nucleic Acids Res.*, 36(10):3455–3462, 2008.