CAMSA: a Tool for Comparative Analysis and Merging of Scaffold Assemblies

S. S. Aganezov[1,2,*] and M. A. Alekseyev[1]

[1] *Department of Mathematics & Computational Biology Institute, The George Washington University, Washington, DC 20052, USA*
http://cblab.org

[2] *Department of Higher Mathematics, ITMO University, St. Petersburg, 197101, Russia*

[*] *E-mail: aganezov@gwu.edu*

Motivation: Despite the recent progress in genome sequencing and assembly, many of the currently available assembled genomes come in a draft form. Such draft genomes consist of a large number of genomic fragments (*scaffolds*), whose positions and orientations along the genome are unknown. While there exists a number of methods for reconstruction of the genome from its scaffolds, utilizing various computational and wet-lab techniques, they often can produce only partial error-prone scaffold assemblies. It therefore becomes important to compare and merge scaffold assemblies produced by different methods, thus combining their advantages and highlighting present conflicts for further investigation. These tasks may be labor intensive if performed manually.

Results: We present CAMSA—a tool for comparative analysis and merging of two or more given scaffold assemblies. The tool (i) creates an extensive report with several comparative quality metrics; (ii) constructs a most consistent combined scaffold assembly; and (iii) provides an interactive framework for a visual comparative analysis of the given assemblies.

Availability: CAMSA is available for download from `http://cblab.org/camsa/`.

Keywords: genome scaffolding, genome assembly, comparative genomics, software

1. Introduction

While genome sequencing technologies are constantly evolving, researchers are still unable to read complete genomic sequences at once from organisms of interest. So, genome reading is usually done in multiple steps, which involve both *in vitro* and *in silico* methods. It starts with reading small genomic fragments, called *reads*, originating from unknown locations in the genome. Modern shotgun sequencing technologies can easily produce millions of reads. The problem then becomes to assemble them into the complete genome. Existing de novo genome assembly algorithms can usually assemble reads into longer genomic fragments, called *contigs*, that are typically interweaved in the genome with highly polymorphic and/or repetitive regions. The next step is to construct *scaffolds*, i.e., sequences of (oriented) contigs along the genome interspaced with gaps. The last but not least step is genome finishing that recovers genomic sequences inside the gaps within the scaffolds.

Unfortunately, the quality of scaffolds (e.g., exposing severe fragmentation) for many genomes makes the finishing step infeasible. As a result, the majority of currently available genomes come in a *draft* form represented by a large number of scaffolds rather than complete chromosomes.[1] This emphasizes the need for improving the assembly quality of genomes by

constructing longer scaffolds from the given ones,[a] which we refer to as the *scaffold assembly problem*.

A number methods have been recently proposed to address the scaffold assembly problem by utilizing various types of additional information and/or *in vitro* experiments. These methods are based on short paired-reads libraries,[2–7] long error-prone reads (such PacBio or MinION reads),[8–12] homology relationship between multiple genomes,[13–15] wet-lab experiments such as the fluorescence *in situ* hybridization (FISH),[16,17] and so on. Depending on the nature and accuracy of utilized information and techniques, assemblies produces by different methods may be incomplete and contain errors, thus deviating from each other. Moreover, some scaffold assemblers can produce only unoriented assemblies, where the (strand-based) orientation of some assembled scaffolds is yet to be determined.

It therefore becomes crucial to determine what parts of different assemblies are consistent with and/or complement each other, and what parts are conflicting with other assemblies (or even within the same assembly). Furthermore, some scaffold assemblies may utilize only a fraction of the input scaffolds (e.g., homology-based assembly methods do not take into account unannotated scaffolds), thus posing a problem of analyzing and comparing assemblies of varying sets of scaffolds. Comparative analysis of scaffold assemblies produced by different methods can help the researchers to combine their advantages and/or highlight potential conflicts for further investigation. These tasks may be labor-intensive if performed manually.

We present CAMSA, a tool for comparative analysis and merging of scaffold assemblies. CAMSA takes as an input two or more assemblies of the same set of scaffolds and generates a comprehensive comparative report for them. The report not only contains multiple numerical characteristics for the input assemblies, but also provides an interactive framework for their visual comparison and analysis. CAMSA also computes a merged assembly, combining the input assemblies into more comprehensive one, which resolves conflicts and determine orientation of unoriented scaffolds in the most confident way. CAMSA is an open source software released under the MIT license.

CAMSA is currently utilized in the study of Anopheles mosquito genomes, where multiple research laboratories (including ours) work on improving the existing assemblies for a set of mosquito species.[18]

2. Methods

Assembly Analysis and Visualization

For the purpose of comparative analysis and visualization of the input scaffold assemblies, CAMSA utilizes the *breakpoint graphs*, the data structure traditionally used for analysis of gene orders across multiple species.[19] We will refer to the breakpoint graph constructed on a set of scaffold assemblies as the *scaffold assembly graph* (SAG).

We start with the case of assemblies with no unoriented scaffolds. Then each assembly A can be viewed as a set of sequences of oriented scaffolds. We represent A as an individual *scaffold*

[a]We remark that contigs can be viewed as scaffolds with no gaps. So, under scaffolds we understand both contigs and scaffolds.

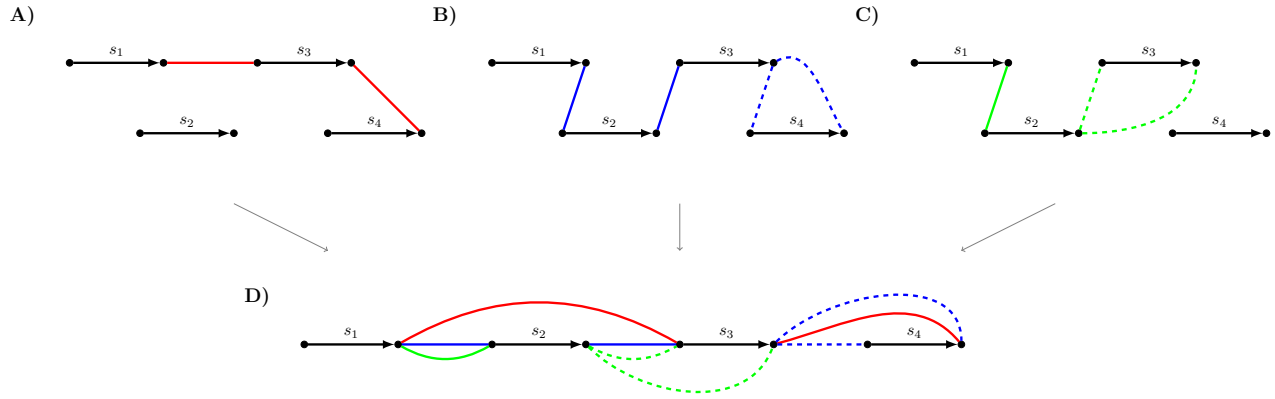


Fig. 1. Individual scaffold assembly graphs for assemblies $A_1 = \{(\overrightarrow{s_1}, \overrightarrow{s_3}, \overleftarrow{s_4})\}$ (red edges), $A_2 = \{(\overrightarrow{s_1}, \overrightarrow{s_2}, \overrightarrow{s_3}, s_4)\}$ (blue edges), and $A_3 = \{(\overrightarrow{s_1}, \overrightarrow{s_2}, s_3), (s_4)\}$ (green edges), and their scaffold assembly graph $\mathrm{SAG}(A_1, A_2, A_3)$. Scaffold edges are colored black. Actual assembly edges are shown as solid, while candidate assembly edges are shown as dashed. **A)** Individual scaffold assembly graph $\mathrm{SAG}(A_1)$. **B)** Individual scaffold assembly graph $\mathrm{SAG}(A_2)$. **C)** Individual scaffold assembly graph $\mathrm{SAG}(A_3)$. **D)** Scaffold assembly graph $\mathrm{SAG}(A_1, A_2, A_3)$.

assembly graph $\mathrm{SAG}(A)$ with two types of edges: directed edges (*scaffold edges*) encoding scaffolds in A, and undirected edges (*assembly edges*) representing scaffold adjacencies and connecting extremities (tails/heads) of the corresponding scaffold edges (Fig. 1A,B,C).

We find it convenient to refer to each assembly edge as an *assembly point*. Equivalently, an assembly point in A can be represented by an ordered pair of oriented scaffolds. We specify the orientation of a scaffold s, either by a sign ($+s$ or $-s$) or by an overhead arrow ($\overrightarrow{s}$ or $\overleftarrow{s}$). For example, $(\overrightarrow{s_1}, \overleftarrow{s_2})$ and $(\overrightarrow{s_2}, \overleftarrow{s_1})$ represent the same assembly point between scaffolds s_1 and s_2 following each other head-to-head. Clearly, any assembly is completely defined by the set of its assembly points.

To construct the scaffold assembly graph $\mathrm{SAG}(A_1, \ldots, A_k)$ of multiple input assemblies $A_1, \ldots, A_k$, we represent them as individual graphs $\mathrm{SAG}(A_1), \ldots, \mathrm{SAG}(A_k)$, where the undirected edges in each $\mathrm{SAG}(A_i)$ are colored into unique color. Then the graph $\mathrm{SAG}(A_1, \ldots, A_k)$ can be viewed as the superposition of individual graphs $\mathrm{SAG}(A_1), \ldots, \mathrm{SAG}(A_k)$ and can be obtained by gluing the identically labeled directed edges. So the graph $\mathrm{SAG}(A_1, \ldots, A_k)$ consists of (directed, labeled) scaffold edges encoding scaffolds and (undirected, unlabeled) assembly edges of k colors encoding assembly points in different input assemblies (Fig. 1D). We will refer to edges of color A_i (i.e., coming from $\mathrm{SAG}(A_i)$) as A_i-edges. The assembly edges connecting the same two vertices x and y form the *multiedge* $\{x, y\}$ in $\mathrm{SAG}(A_1, \ldots, A_k)$. The *multicolor* of $\{x, y\}$ is defined as the union of the colors of individual edges connecting x and y.

We define the (ordinary) *degree* $\mathrm{odeg}(x)$ of a vertex x in $\mathrm{SAG}(A_1, \ldots, A_k)$ as the number of assembly edges incident to x. We distinguish it from the *multidegree* $\mathrm{mdeg}(x)$ defined as the number of adjacent vertices that are connected to x with assembly edges.

When all assemblies $A_1, \ldots, A_k$ agree on a particular assembly point $\{x, y\}$, the graph $\mathrm{SAG}(A_1, \ldots, A_k)$ contains a multi-edge $\{x, y\}$ composed of edges of all k different colors. In other words, both vertices x and y in this case have degree k and multidegree 1. For a vertex z

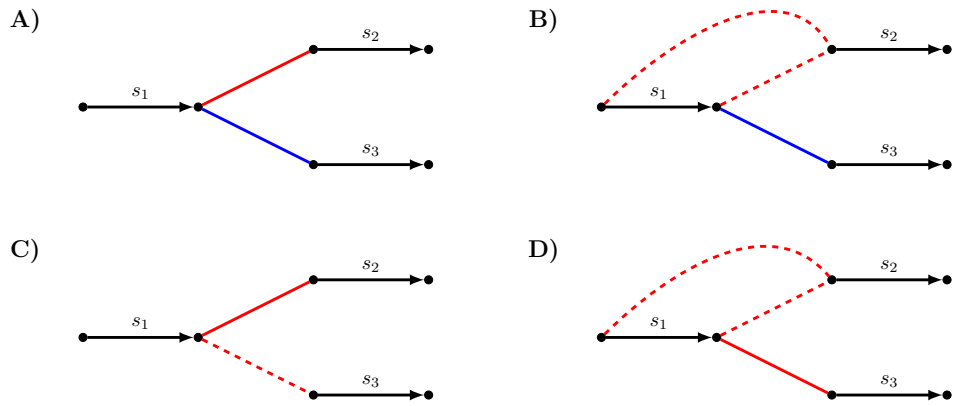


Fig. 2. Illustration of various conflicts between assembly points of assemblies A_1 (red edges) and A_2 (blue edges). **A)** Assembly points $(\overrightarrow{s_1}, \overrightarrow{s_2})$ from assembly A_1 and $(\overrightarrow{s_1}, \overrightarrow{s_3})$ from assembly A_2 are out-conflicting. **B)** Assembly points $(s_1, \overrightarrow{s_2})$ from A_1 and $(\overrightarrow{s_1}, \overrightarrow{s_2})$ from A_2 are out-semiconflicting. **C)** Assembly points $(\overrightarrow{s_1}, \overrightarrow{s_2})$ and $(\overrightarrow{s_2}, \overrightarrow{s_3})$ both from A_1 are in-conflicting. **D)** assembly points $(s_1, \overrightarrow{s_2})$ and $(\overrightarrow{s_1}, \overrightarrow{s_2})$ both from A_1 are *in-semiconflicting*.

in $\mathrm{SAG}(A_1, \ldots, A_k)$, $\mathrm{odeg}(z) \neq k$ or $\mathrm{mdeg}(z) \neq 1$ indicate some type of inconsistency between the assemblies.

We classify an individual assembly points $\{x, y\}$ as follows. Let S be the multicolor of the multiedge $\{x, y\}$ in $\mathrm{SAG}(A_1, \ldots, A_k)$.

- **unique** if $|S| = 1$, i.e., the assembly point $\{x, y\}$ is present only in a single assembly;
- **in-conflicting** within assembly $A \in S$ if x or y is incident any other A-edges besides $\{x, y\}$ (e.g., Fig. 2C);
- **out-conflicting** if there exist two distinct assemblies A and B such that A contains $\{x, y\}$ (i.e., $A \in S$), and B contains $\{x, z\}$ with $z \neq y$ or $\{y, z\}$ with $z \neq x$ (e.g., Fig. 2A).

Dealing with Unoriented Scaffolds

While conventional multiple breakpoint graphs are constructed for sequences of oriented genes, in CAMSA we extend scaffold assembly graphs to support assemblies that may include oriented as well as unoriented scaffolds.

In addition to (oriented) assembly points formed by pairs of oriented scaffolds, we now consider semi-oriented and unoriented assembly points.

A *semi-oriented* assembly point represents an adjacency between an oriented scaffold and an unoriented one. For example, $(\overleftarrow{s_1}, s_2)$ and $(s_2, \overrightarrow{s_1})$ denote the same semi-oriented assembly point, where scaffold s_1 is oriented and s_2 is not (as emphasized by a missing overhead arrow). Similarly, an *unoriented* assembly point represents an adjacency between two unoriented scaffolds. For example, (s_1, s_2) and (s_2, s_1) denote the same unoriented assembly point between unoriented scaffolds s_1 and s_2.

We define a *realization* of an assembly point p as any oriented assembly point that can be obtained from p by orienting unoriented scaffolds. We denote the set of orientations of p as $R(p)$. If p is oriented, then it has a single realization equal to p itself (i.e., $R(p) = \{p\}$); if p is

semi-oriented, then it has two realizations (i.e., $|R(p)| = 2$); and if p is unoriented, then it has four realizations (i.e., $|R(p)| = 4$). For example,

$$R((s_1, s_2)) = \{(\overleftarrow{s_1}, \overleftarrow{s_2}), (\overleftarrow{s_1}, \overrightarrow{s_2}), (\overrightarrow{s_1}, \overleftarrow{s_2}), (\overrightarrow{s_1}, \overrightarrow{s_2})\}.$$

In the scaffold assembly graph, we add assembly edges encoding all realizations of semi-/un- oriented assembly points and refer to such edges as *candidate*, in contrast to *actual* assembly edges encoding oriented assembly points.

We extend the in-/out- conflicting classification to semi-oriented and unoriented assembly points as follows. An assembly point is *in-/out- conflicting* if all its realizations are such, except that we do not consider two realizations as in-conflicting to each other. Similarly, an assembly point is *in-/out- semiconflicting* if some but not all of its realizations are in-/out- conflicting (e.g., Fig. 2B,D illustrate pairs of out-semiconflicting and in-semiconflicting assembly points, respectively).

Merging Assemblies

CAMSA can resolve conflicts in the input assemblies by merging them into a single (not self-conflicting) *merged assembly* that is most consistent with the input ones. The merged assembly is also used to determine orientation of (some) unoriented scaffolds in one input assemblies that is most confident and/or consistent with other input assemblies. In other words, the merged assembly helps to identify realizations of (some) semi-/un- oriented assembly points that are most consistent with other assemblies. Namely, for each semi-/un- oriented assembly point, the merged assembly contains either only one or none of its realizations; and in the former case, the included realization defines the most confident orientation of the corresponding unoriented scaffolds.

Assembly merging performed by CAMSA is based on how often each assembly point appears in the input assemblies as well as on the (optional) confidence of each such appearance. Namely, for each assembly point p in an input assembly A, CAMSA allows to specify the *confidence weight* $\mathrm{CW}_A(p)$ from the interval $[0, 1]$, which is then assigned to the corresponding assembly edge(s) (Fig. 3A). The confidence weights are expected to reflect the confidence level of the assembly methods in what they report as scaffold adjacencies (e.g., heuristic methods should probably have smaller confidence as compared to more reliable wet-lab techniques). By default, all actual assembly edges have the confidence weight equal 1, and all candidate assembly edges have weight 0.75 (these default values can be overwritten by the user).

For any oriented assembly B (viewed as a set of oriented assembly points), we define the *consistency score* $\mathrm{CS}_B(A)$ of an input assembly A with respect to B as $\mathrm{CS}_B(A) = \sum_{p \in B} \overline{\mathrm{CW}}_A(p)$, where

$$\overline{\mathrm{CW}}_A(p) = \begin{cases} 0, & \text{if } \forall\ x \in A\ :\ p \notin R(x); \\ \mathrm{CW}_A(x), & \text{if } \exists\ x \in A\ :\ p \in R(x). \end{cases}$$

We pose the *assembly merging problem* (AMP) as follows.

Problem 2.1 (Assembly Merging Problem, AMP). *Given assemblies $A_1, \ldots, A_k$ of the same set of scaffolds S, find an assembly M of S containing only oriented assembly points such that*

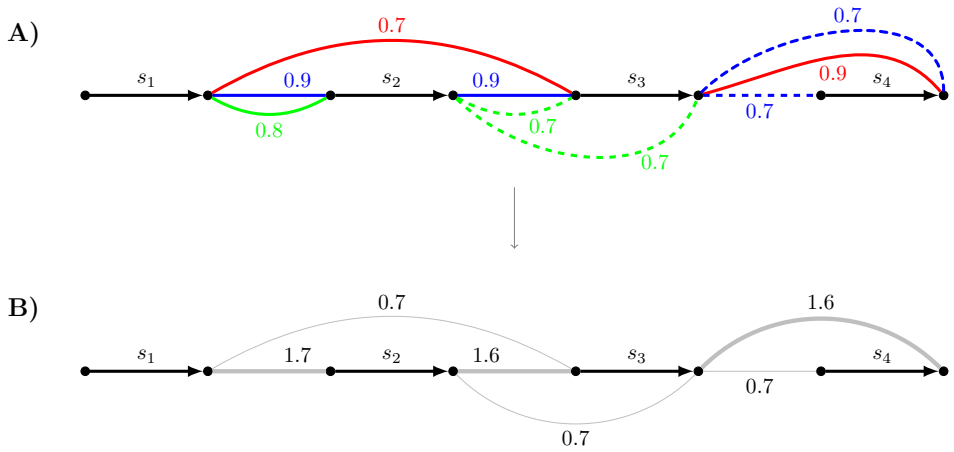


Fig. 3. A) Scaffold assembly graph $\mathrm{SAG}(A_1, A_2, A_3)$, where $A_1 = \{(\overrightarrow{s_1}, \overrightarrow{s_3}, 0.7), (\overrightarrow{s_3}, \overleftarrow{s_4}, 0.9)\}$ (red edges), $A_2 = \{(\overrightarrow{s_1}, \overrightarrow{s_2}, 0.9), (\overrightarrow{s_2}, \overrightarrow{s_3}, 0.9), (\overrightarrow{s_3}, s_4, 0.7)\}$ (blue), and $A_3 = \{(\overrightarrow{s_1}, \overrightarrow{s_2}, 0.8), (\overrightarrow{s_2}, s_3, 0.7)\}$ (green), where the numerical value in each assembly point represents its confidence weights. B) Merged scaffold assembly graph $\mathrm{MSAG}(A_1, A_2, A_3)$, obtained from $\mathrm{SAG}(A_1, A_2, A_3)$ by collapsing assembly multi-edges. Scaffold edges combined with bold gray edges represent the merged assembly $M = \{(\overrightarrow{s_1}, \overrightarrow{s_2}, 1.7), (\overrightarrow{s_2}, \overrightarrow{s_3}, 1.6), (\overrightarrow{s_3}, \overleftarrow{s_4}, 1.6)\}$, computed by CAMSA, which solves the AMP problem with respect to input assemblies A_1, A_2 and A_3.

(i) M *is not self-conflicting (i.e., does not contain any in-conflicting assembly points);*
(ii) $\sum_{i=1}^{k} \mathrm{CS}_M(A_i)$ *is maximized;*
(iii) for every assembly point $p \in A_1 \cup \cdots \cup A_k$*, at most one of its representations is present in* M *(i.e.,* $|M \cap R(p)| \leq 1$*).*

For a solution M to the AMP, the condition (i) implies that the assembly edges in $\mathrm{SAG}(M)$ form a matching. Furthermore, M is assumed to correspond to the genome, which may be a subject to additional constraints such as having all chromosomes linear (e.g., for vertebrate genomes) or having a single chromosome (e.g., for bacterial genomes). These constraints are translated for M as the absence in $\mathrm{SAG}(M)$ of cycles formed by alternating assembly and scaffold edges (for a unichromosomal circular genome, such a cycle can be present in $\mathrm{SAG}(M)$ only if it includes all scaffold edges).

To address the AMP, we first construct the (weighted) *merged scaffold assembly graph* $\mathrm{MSAG}(A_1, \ldots, A_k)$ from $\mathrm{SAG}(A_1, \ldots, A_k)$ by replacing each assembly multi-edge with an ordinary assembly edge of the weight equal the total weight of the corresponding multi-edge (Figure 3). So, $\mathrm{MSAG}(A_1, \ldots, A_k)$ is the graph with two types of edges: unweighted directed scaffolds edges and weighted undirected assembly edges. The AMP is then can be reformulated as the following *restricted maximum matching problem* (RMMP) on the graph $G = \mathrm{MSAG}(A_1, \ldots, A_k)$:

Problem 2.2 (Restricted Maximum Matching Problem, RMMP). *Given a merged scaffold assembly graph* G*, find a subset* M *of assembly edges in* G *such that*

(i) M *is a matching;*
(ii) M *has maximum weight;*
(iii) there are no cycles in $\mathrm{SAG}(M)$*.*

Let M be a solution to the RMMP. Then the graph $\mathrm{SAG}(M)$ consists of scaffold edges forming a perfect matching and assembly edges from M forming a (possibly non-perfect) matching by the condition (i). Thus $\mathrm{SAG}(M)$ is formed by collection of paths and cycles, whose edges alternate between scaffold and assembly edges. Furthermore, by the condition (iii), $\mathrm{SAG}(M)$ consists entirely of alternating paths. A similar optimization problem, where the number of paths and the number cycles in the resulting $\mathrm{SAG}(M)$ are fixed, is known to be NP-complete,[20] leaving a little hope for the RMMP to have a polynomial-time solution. Instead, CAMSA employs two merging heuristic solutions building upon the previously proposed algorithms[20,21] as we describe below in this section.

Progressive merging heuristics. For a given merged scaffold assembly G, this strategy starts with the graph H consisting of scaffold edges from G and then iteratively enriches H with assembly edges so that no cycles are created in H. At any stage of this process, H is considered as a collection of alternating paths, some of which are merged into a longer path by adding a corresponding assembly edge. The paths to merge are selected based on the confidence weight of their linking assembly edge. The final graph H constructed this way defines M as the set of assembly edges in H (and so $\mathrm{SAG}(M) = H$).

Maximum matching heuristics. For a given merged scaffold assembly G, this strategy constructs M' by computing the maximum weighted matching formed by assembly edges of G. Then it looks for cycles in $\mathrm{SAG}(M')$ (notice that all cycles in $\mathrm{SAG}(M')$ are vertex-disjoint) and removes an assembly edge of the lowest confidence weight from each cycle. These edges are also removed from M' to form M so that $\mathrm{SAG}(M)$ consists entirely of alternating paths.

We remark that before solving the RMMP for $G = \mathrm{MSAG}(A_1, \ldots, A_k)$, CAMSA allows to remove assembly edges from G that have weight smaller than the *weight threshold* specified by the user (by default, this threshold is set to 0, i.e., no edges are removed). The removal of small-weighted assembly edges may be desirable if one wants to restrict attention only to assembly points of certain confidence level (e.g., assembly points coming either from individual highly-reliable assemblies, or as a consensus from multiple assemblies). When such removal of low-confidence edges is performed, it is important to do so before (not after) solving the RMMP, since otherwise these edges may introduce a bias for inclusion of high-confidence edges into the merged assembly M.

3. Structure of CAMSA Report

The results of comparative analysis and assembly merging performed by CAMSA are presented to the user in the form of interactive *report*. The report is generated in a JavaScript-powered HTML file, readily accessible for viewing/working in any modern Internet browser (for locally generated reports, Internet connection is not required). Many of the report sections are also available in the form of text files, making them accessible for machine processing. All tables in the report are powered by the DataTables JavaScript library,[22] which provides flexible and dynamic filtering, sorting, and searching capabilities.

The first section of the CAMSA report presents aggregated characteristics of each input

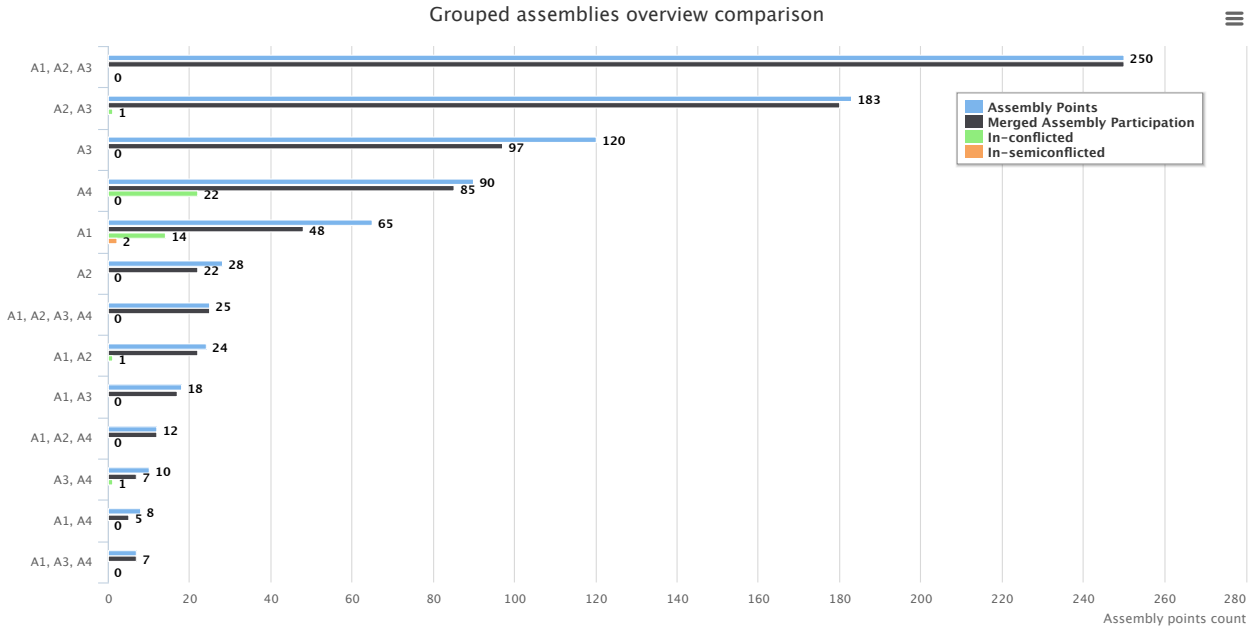


Fig. 4. The second section of the CAMSA report. The following characteristics are computed for different subsets of the input assemblies $A1$, $A2$, $A3$, and $A4$: the number of assembly points that are unique to the observed subset; the number of assembly points that participate (by at least one of their realizations) in the merged assembly; the number of in-conflicting assembly points; the number of in-semiconflicting assembly points.

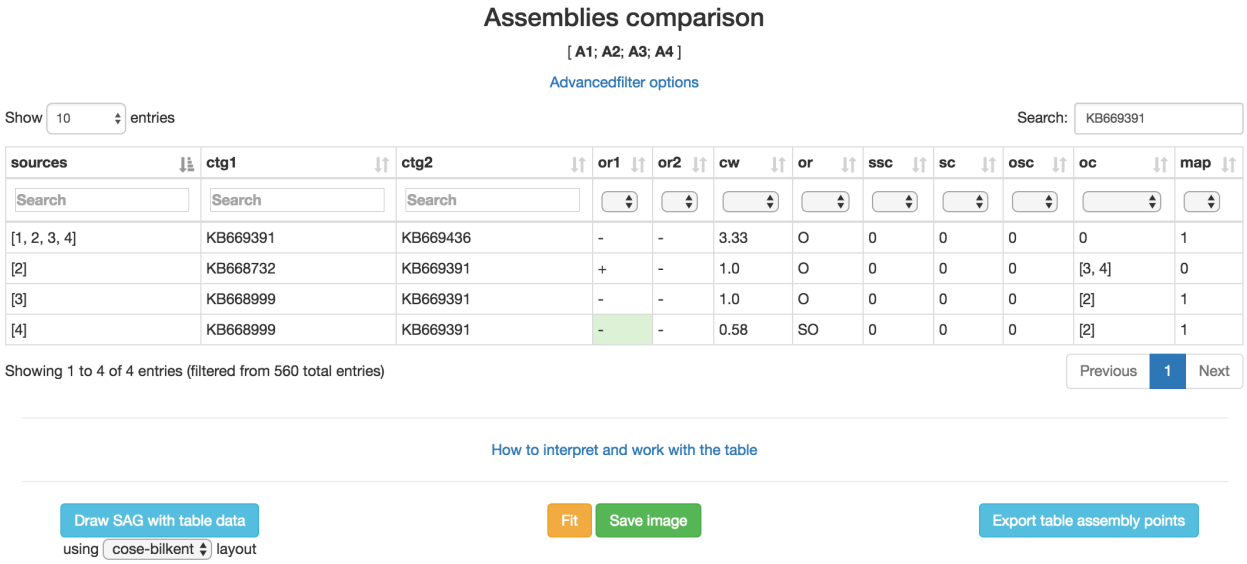
assemblies as compared to the others:

(i) the number of oriented, semi-oriented, and unoriented assembly points;
(ii) the number of in-/out- conflicting assembly points;
(iii) the number of in-/out- semiconflicting assembly points;
(iv) the number of nonconflicting assembly points;
(v) the number of assembly points that participate in the merged assembly.

The second section of the CAMSA report focuses on consistency across various subsets of input assemblies (Fig. 4). For each subset, characteristics similar to the ones in the first section are computed, but the values here are aggregated over all assemblies in the subset. Such statistics eliminates the need of running CAMSA separately on any assemblies subsets and allows the user to easily identify groups of assemblies that agree/conflict among themselves the most. We remark that each assembly point is counted only once, for the subset of assemblies that contain this assembly point (but not for any smaller subset of them). As the the number of all possible subsets of input assemblies grows exponentially, CAMSA provides a way for user to specify the maximum number of such subsets (sorted in descending order of accounted assembly points) to be displayed.

The third section of the CAMSA report provides statistics for each assembly point within each assembly. Extensive interactive filtering allows the user to select assembly points of interest, as well as to export the filtered results, creating problem- / region- / fragment- focused analysis pipelines. We remark that statistical characteristics (e.g., whether an assembly point

A)



B)

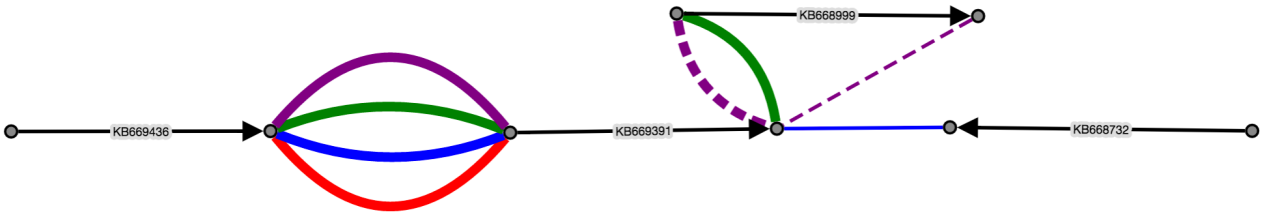


Fig. 5. The fourth section of the CAMSA report. **A)** Table-wide filtration that retain information only about assembly points involving scaffold `KB669391`. Each assembly point shown in the table is unique; the input assemblies containing a particular assembly point are listed in the *sources* column (e.g., the assembly point ($\overleftarrow{\text{KB669391}}, \overleftarrow{\text{KB669436}}$) is present in all four input assemblies). The table also provides various statistics and allows filtering over them. **B)** A subgraph of the scaffold assembly graph $\text{SAG}(A_1, A_2, A_3, A_4)$ induced by the assembly points involving scaffold `KB669391`.

is in-/out- conflicting or in-/out- semiconflicting) are computed with respect to all of the input assemblies.

The fourth section of the CAMSA report provides statistics for each assembly point aggregated over all of the input assemblies (Fig. 5). In contrast to the third section, each assembly point is shown here exactly once, and the *sources* column shows the set of assemblies where this assembly point is present. Again, CAMSA provides extensive filtering to enable a focused analysis of assembly points of interest. The result of assembly points filtration can further be exported in the same format, which is utilized for CAMSA input files (i.e., tab-separated list of distinct assembly points).

Besides the text-based representation and export, the CAMSA report also provides an interactive visualization and further graphical export of assembly points in the form of the scaffold assembly graph. A vector-based interactive graph visualization is created using the

Cytoscape.js library.[23] This representation is dynamic with respect to the graph layout and supports filtration of graph components. We allow the users to choose among several Cytoscape.js graph layouts (default from Ref. 24). The time required for graph visualization heavily depends on the chosen layout and the underlying graph complexity. In cases when visualization inside the report takes too long, we provide a workaround: assembly points can be exported in a text format and then converted into DOT-formatted file describing the corresponding graph, using on of the utility scripts, distributed as a part of CAMSA. At any point the current graph layout can be exported from the report in the PNG format.

4. Evaluation

We evaluate CAMSA performance on several genomic datasets with varying number of input assemblies and assembly points. All evaluations were performed on a *MacBook Pro11* with *Intel Core i7* processor (2.8 GHz, 2 cores) and 16 GB of RAM. We evaluate the running time for the CAMSA analysis and report generation along with both progressive (PM) and maximal matching (MM) merging heuristics, using NetworkX library.[25]

First evaluation of CAMSA is based on its current utilization in the Anopheles Genome Cluster Consortium (AGCC) project for completing genomes of multiple *Anopheles* species.[18] Presence of complex genomic repeats and polyploid origin of these genome makes their complete assembly extremely challenging. Many of the currently assembled Anopheles genomes appear in the form of hundreds or even thousands scaffolds. Within the AGCC project, several methods have been deployed to address the scaffold assembly problem for these genomes. The preliminary CAMSA analysis revealed that these methods well complement each other. For evaluation purposes, we run CAMSA on four distinct assemblies of the same 834 *A. funestus* scaffolds with the total of 1,040 input assembly points. The running time for CAMSA was about 1 and 3 seconds when it uses the PM and MM heuristics, respectively.

The second evaluation of CAMSA is based on assembly of actual contigs (rather than scaffolds) in three genomic datasets by the following scaffolders: ScaffMatch,[26] SOAPdenovo2,[5] and SGA.[27] As an input to these scaffolders, we provided the contigs constructed by ALLPATHS-LG[28] and a short-jump library from the GAGE project.[29] Since scaffolders typically report their results in form of genomic sequences (rather than sequences of contigs) representing scaffolds, we have developed a script that maps the contigs onto the scaffolds (using NUCmer[30] with at least 90% query coverage) and identifies contig adjacencies with the scaffolds. We provided the identified contig adjacencies as input assembly points for CAMSA. The results shown in Table 1 demonstrate that CAMSA can effectively handle large input datasets with thousands of assembly points.

5. Conclusions

CAMSA addresses the current deficiency of tools for automated comparison and merging of multiple assemblies of the same set scaffolds. Since there exist numerous methods and techniques for scaffold assembly, identifying similarities and dissimilarities across assemblies produced by different methods is beneficial both for the developers of scaffold assembly algorithms and for the researchers focused on improving draft assemblies of specific organisms.

Table 1. Evaluation of CAMSA on scaffold assemblies produced by ScaffMatch, SOAPdenovo2, and SGA on three datasets.

Dataset	# contigs	Total # AP	PM (sec)	MM (sec)
Staphylococcus aureus	67	63	< 1	< 1
Rhodobacter sphaeroides	208	185	< 1	< 1
Human Chromosome 14	4,722	15,120	48	52

We remark that CAMSA expects as an input a list of assembly points, while many conventional scaffolding tools report their results in the form of genomic sequences. This inspired us to develop a set of convertion scripts that automate the input preparation for CAMSA (e.g., see Section 4) and are included the CAMSA distribution.

CAMSA is an active project with evolving functionality. For example, we plan to add more information into the analysis pipeline (e.g., the gap sizes in assembly points, merged assembly length estimation, and so on). We further plan to enrich the graph-based analysis in CAMSA with various pattern matching techniques, enabling a better classification of assembly conflicts based on their origin (e.g., conflicting scaffold orders, wrong orientation of scaffolds, or different resolution of assemblies). We also work on improving the assembly merging algorithms in CAMSA to allow better accuracy and imposing specific genomic features (e.g., linear vs. circular chromosomes) in the assembled genome.

Acknowledgements

The authors thank the members of the Anopheles Genome Cluster Consortium for their feedback and helpful suggestions during the development of CAMSA.

The work is supported by the National Science Foundation under the grant No. IIS-1462107.

References

1. T. Reddy, A. D. Thomas, D. Stamatis, J. Bertsch, M. Isbandi, J. Jansson, J. Mallajosyula, I. Pagani, E. A. Lobos and N. C. Kyrpides, *Nucleic Acids Research* , p. gku950 (2014).
2. J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones and I. Birol, *Genome Research* **19**, 1117 (2009).
3. S. Koren, T. J. Treangen and M. Pop, *Bioinformatics* **27**, 2964 (2011).
4. A. A. Gritsenko, J. F. Nijkamp, M. J. Reinders and D. de Ridder, *Bioinformatics* **28**, 1429 (2012).
5. R. Luo, B. Liu, Y. Xie, Z. Li, W. Huang, J. Yuan, G. He, Y. Chen, Q. Pan, Y. Liu *et al.*, *GigaScience* **1**, p. 1 (2012).
6. A. Dayarian, T. P. Michael and A. M. Sengupta, *BMC Bioinformatics* **11**, p. 1 (2010).
7. M. Boetzer, C. V. Henkel, H. J. Jansen, D. Butler and W. Pirovano, *Bioinformatics* **27**, 578 (2011).
8. R. L. Warren, C. Yang, B. P. Vandervalk, B. Behsaz, A. Lagman, S. J. Jones and I. Birol, *GigaScience* **4**, p. 1 (2015).
9. 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 *et al.*, *Journal of Computational Biology* **19**, 455 (2012).

10. A. Bashir, A. A. Klammer, W. P. Robins, C.-S. Chin, D. Webster, E. Paxinos, D. Hsu, M. Ashby, S. Wang, P. Peluso *et al.*, *Nature Biotechnology* **30**, 701 (2012).
11. M. Boetzer and W. Pirovano, *BMC Bioinformatics* **15**, p. 1 (2014).
12. K.-K. Lam, K. LaButti, A. Khalak and D. Tse, *Bioinformatics* **31**, 3207 (2015).
13. L. Assour and S. Emrich, Multi-genome synteny for assembly improvement, in *Proceedings of 7th International Conference on Bioinformatics and Computational Biology*, 2015.
14. S. Aganezov and M. A. Alekseyev, Multi-Genome Scaffold Co-Assembly Based on the Analysis of Gene Orders and Genomic Repeats, in *Proceedings of the 12th International Symposium on Bioinformatics Research and Applications (ISBRA)*, eds. A. Bourgeois *et al.*, Lecture Notes in Computer Science, Vol. 9683(2016).
15. Y. Anselmetti, V. Berry, C. Chauve, A. Chateau, E. Tannier and S. Bérard, *BMC Genomics* **16**, 1 (2015).
16. G. T. Rudkin and B. Stollar (1977).
17. M. R. Speicher and N. P. Carter, *Nature Reviews Genetics* **6**, 782 (2005).
18. D. E. Neafsey, R. M. Waterhouse, M. R. Abai, S. S. Aganezov, M. A. Alekseyev *et al.*, *Science* **347**, p. 1258522 (2015).
19. P. Avdeyev, S. Jiang, S. Aganezov, F. Hu and M. A. Alekseyev, *Journal of Computational Biology* **23**, 1 (2016).
20. A. Chateau and R. Giroudeau, *Theoretical Computer Science* **595**, 92 (2015).
21. S. Moran, I. Newman and Y. Wolfstahl, *Networks* **20**, 55 (1990).
22. A. Jardine, DataTables JavaScript / JQuery library `https://datatables.net`, (2011), Accessed: 2016-06-13.
23. M. Franz, C. T. Lopes, G. Huck, Y. Dong, O. Sumer and G. D. Bader, *Bioinformatics* **32**, 309 (2016).
24. U. Dogrusoz, E. Giral, A. Cetintas, A. Civril and E. Demir, *Information Sciences* **179**, 980 (2009).
25. A. A. Hagberg, D. A. Schult and P. J. Swart, Exploring network structure, dynamics, and function using NetworkX, in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, (Pasadena, CA USA, 2008).
26. I. Mandric and A. Zelikovsky, *Bioinformatics* , p. btv211 (2015).
27. J. T. Simpson and R. Durbin, *Genome Research* **22**, 549 (2012).
28. S. Gnerre, I. MacCallum, D. Przybylski, F. J. Ribeiro, J. N. Burton, B. J. Walker, T. Sharpe, G. Hall, T. P. Shea, S. Sykes *et al.*, *Proceedings of the National Academy of Sciences* **108**, 1513 (2011).
29. S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts *et al.*, *Genome Research* **22**, 557 (2012).
30. S. Kurtz, A. Phillippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu and S. L. Salzberg, *Genome Biology* **5**, p. 1 (2004).