Innovation, Sciences et Développement économique Canada
Office de la Propriété Intellectuelle du Canada

Innovation, Science and Economic Development Canada
Canadian Intellectual Property Office

CA 2964349 C 2023/03/21

(11)(21) 2 964 349

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(86) **Date de dépôt PCT/PCT Filing Date:** 2015/10/07
(87) **Date publication PCT/PCT Publication Date:** 2016/04/21
(45) **Date de délivrance/Issue Date:** 2023/03/21
(85) **Entrée phase nationale/National Entry:** 2017/04/11
(86) **N° demande PCT/PCT Application No.:** US 2015/054461
(87) **N° publication PCT/PCT Publication No.:** 2016/060910
(30) **Priorité/Priority:** 2014/10/14 (US62/063,545)

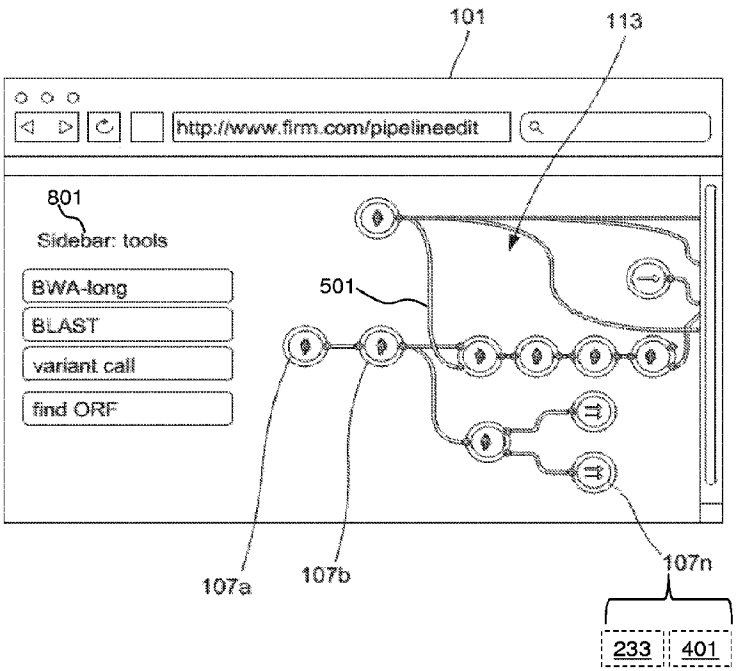
(51) **Cl.Int./Int.Cl.** ***G16B 50/00*** (2019.01),
C12Q 1/68 (2018.01), ***G16B 20/00*** (2019.01),
G16B 30/00 (2019.01), ***G16B 40/00*** (2019.01)

(72) **Inventeurs/Inventors:**
TIJANIC, NEBOJSA, US;
STOJANOVIC, LUKA, US;
COHADAREVIC, DAMIR, US;
IVKOVIC, SINISA, US

(73) **Propriétaire/Owner:**
SEVEN BRIDGES GENOMICS INC., US

(74) **Agent:** DEETH WILLIAMS WALL LLP

(54) **Titre : SYSTEMES ET PROCEDES POUR OUTILS INTELLIGENTS DANS DES PIPELINES DE TRAITEMENT DE SEQUENCES**
(54) **Title: SYSTEMS AND METHODS FOR SMART TOOLS IN SEQUENCE PIPELINES**



(57) **Abrégé/Abstract:**
The invention relates to bioinformatics pipelines and wrapper scripts that call executables in those pipelines and that also identify beneficial changes to the pipelines. A tool in a pipeline has a smart wrapper that can cause the tool to analyze the sequence data it receives but that can also select a change to the pipeline when circumstances warrant. In certain aspects, the invention provides a system for genomic analysis. The system includes a processor coupled to a non- transitory memory. The system is operable to present to a user a plurality of genomic tools organized into a pipeline. At least a first one of the tools comprises an executable and a wrapper script. The system can receive instructions from the user and sequence data- instructions that call for the sequence data to be analyzed by the pipeline- and select, using the wrapper script, a change to the pipeline.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
21 April 2016 (21.04.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/060910 A1

(51) International Patent Classification:
G06F 19/22 (2011.01)

(21) International Application Number:
PCT/US2015/054461

(22) International Filing Date:
7 October 2015 (07.10.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/063,545 14 October 2014 (14.10.2014) US

(71) Applicant: **SEVEN BRIDGES GENOMICS INC.** [US/US]; 1 Broadway, 14th Floor, Cambridge, MA 02142 (US).

(72) Inventors: **TANIC, Nebojsa**; 1 Broadway, 14th Floor, Cambridge, MA 02142 (US). **STOJANOVI, Luka**; 1 Broadway, 14th Floor, Cambridge, MA 02142 (US). **COHADAREVIC, Damir**; 1 Broadway, 14th Floor, Cambridge, MA 02142 (US). **IVKOVIC, Sinisa**; 1 Broadway, 14th Floor, Cambridge, MA 02142 (US).

(74) Agents: **MEYERS, Thomas C.** et al.; Brown Rudnick LLP, One Financial Center, Boston, MA 02111 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

WO 2016/060910 A1

(54) Title: SYSTEMS AND METHODS FOR SMART TOOLS IN SEQUENCE PIPELINES

(57) Abstract: The invention relates to bioinformatics pipelines and wrapper scripts that call executables in those pipelines and that also identify beneficial changes to the pipelines. A tool in a pipeline has a smart wrapper that can cause the tool to analyze the sequence data it receives but that can also select a change to the pipeline when circumstances warrant. In certain aspects, the invention provides a system for genomic analysis. The system includes a processor coupled to a non- transitory memory. The system is operable to present to a user a plurality of genomic tools organized into a pipeline. At least a first one of the tools comprises an executable and a wrapper script. The system can receive instructions from the user and sequence data— instructions that call for the sequence data to be analyzed by the pipeline— and select, using the wrapper script, a change to the pipeline.

SYSTEMS AND METHODS FOR SMART TOOLS IN SEQUENCE PIPELINES

Cross-Reference to Related Application

This application claims priority to, and the benefit of, U.S. Provisional Patent Application Serial No. 62/063,545, filed October 14, 2014.

Field of the Invention

The invention generally relates to bioinformatics pipelines and to wrapper scripts that call executables in those pipelines and that also identify beneficial changes to the pipelines.

Background

Examining a person's genes can reveal if that person has a genetic disease or even if he or she is a latent carrier of a disease, at risk of passing the disease on to his or her children. The information is the persons' genes can be revealed by DNA sequencing. The DNA sequencing technologies known as next-generation sequencing (NGS) are capable of sequencing an entire human genome in under a day and for under $1,000. See Clark, Illumina announces landmark $1,000 human genome sequencing, Wired, 15 January 2014. The output of NGS instruments typically includes many short sequence reads that must be assembled together and compared to known genetic information to meaningfully determine a person's genetic information.

This assembly and analysis is not a trivial task, and different computer program tools exist that perform various pieces of the assembly and analysis job. There are computer platforms that provide a graphical user interface (GUI) that can be used by a researcher or medical professional to assemble genomic analysis tools into pipelines that perform complex analytical tasks on sequence data. See, e.g., Torri, Next generation sequence analysis and computational genomics using graphical pipeline workflows, Genes (Basel) 3(3):545-75 (2012). However, these pipeline editors require the user to have mastered the intricacies of the underlying tools. If

Date Reçue/Date Received 2022-02-04

the user wants sequence reads to be aligned to a reference genome, for example, the user must be familiar with the myriad alignment tools such as MAQ, Burrows-Wheeler Aligner, SHRiMP, ZOOM, BFAST, MOSAIK, PERM, MUMmer, PROmer, BLAT, SOAP2, ELAND, RTG Investigator, Novoalign, Exonerate, Clustal Omega, ClustalW, ClustalX, and FASTA, to name a few. Additionally, the user must have a meaningful understanding of the sequence file (e.g., VCF, FASTA, FASTQ, SAM, GenBank, Nexus, EMBL, GCG, SwissProt, PIR, phylip, msf, hennig86, jackknifer) and know which is which and at what points one needs to be converted to another, and what formats are the default inputs and outputs of each tool within a pipeline. Due to the complexities involved, working within a graphical pipeline editor does not solve all the challenges in assembling and analyzing sequence data. Data files may be passed along in the wrong format, causing a program to throw an error and abort the pipeline. In some cases, the tool selected to do a job will be a poor choice and will not work efficiently with the kind of data passed to it or—worse yet—will provide a substantively incorrect output. For example, an inconsistency between the choice of tool, the sequence data, the instructions provided by the user, and the user's expectation may actually cause the pipeline to not provide the correct result and potentially miss an important mutation.

Summary

The invention provides pipelines in which a tool has a smart wrapper that can cause the tool to analyze the sequence data it receives but that can also select a change to the pipeline when circumstances warrant. For example, the smart wrapper can detect an inconsistency between the input data and the tool (e.g., wrong format) and can cause the pipeline to fix the input data before running the tool. Alternatively, the smart wrapper can detect an inconsistency between the input data and the tool and call an alternative second tool that accepts the input data format to perform the analysis. In another example, a smart wrapper can detect that a proposed analysis calls for some additional resource and can fetch that resource (e.g., can fetch a file containing a reference genome for variant calling). Smart wrappers can recover from pipeline errors by reading an error message and making the appropriate correction (e.g., a DNA sequence file that includes an "E" in the sequence data may cause a program to stop and issue an error; the smart wrapper could re-code the "E" to "N"). Since the smart wrapper is capable of dealing with errors from the tools or inconsistencies among the data, the tools, and the instructions, pipelines that include tools with

smart wrappers will avoid mistakes and run to completion to provide the user with an analytical result that is correct and consistent with the user's expectations. Thus sequence assembly and analysis will produce the desired results and be successful, meaning that genetic sequence analysis can be adopted widely in medicine and research and used to solve scientific and medical problems.

In certain aspects, the invention provides a system for genomic analysis. The system includes a processor coupled to a non-transitory memory. The system is operable to present to a user a plurality of genomic tools organized into a pipeline. At least a first one of the tools comprises an executable and a wrapper script. The system can receive, from the user, instructions that call for the sequence data to be analyzed by the pipeline and sequence data and select, using the wrapper script, a change to the pipeline. The wrapper script may analyze the sequence data and select the change based on a feature of the sequence data. The change to the pipeline may include execution of an alternative executable instead of the executable. The wrapper script may select the change in response to an error produced by one of the tools. The wrapper script can recommend the change to the user and allows the user to accept the recommendation. In some embodiments, the wrapper script further performs the change to the pipeline.

In certain embodiments, he wrapper script selects to not analyze the sequence data with the executable. The wrapper script may recommend that the user use a second tool instead of the first one of the tools. For example, the executable may include a sequence alignment program and the change to the pipeline includes an alternative sequence alignment program.

The selected change may include a request for additional resources and the wrapper script can make the request. The requested additional resource may include using the system for: retrieving a data file not provided by the user and not included in the sequence data; retrieving data from a URL; retrieving a matrix of probabilities; calling for a first tool in the pipeline to generate ancillary data from the sequence data to be used by a subsequent tool in the pipeline when the subsequent tool analyzes the sequence data; requesting additional computing power; requesting additional computer processors; requesting one or more virtual machines; and requesting additional storage space.

The instructions may include at least one flag that establishes a value for a parameter, and the smart wrapper selects the change by changing the flag to establish a different value for the

parameter. The wrapper script can add a flag to the instructions that sends a parameter to the executable, wherein the parameter controls how the executable analyzes the sequence data.

In some embodiments, the wrapper script selects the change to the pipeline by receiving an error from the executable, making an adjustment that avoids the error and re-running the executable.

The wrapper script can be used to detect an inconsistency between the instructions and the executable, between the instructions and the sequence data, or between the sequence data and the executable. The wrapper script may cause the system to: prompt the user for additional data; prompt the user to accept the selected change; inform the user of the selected change; or take other action.

Aspects of the invention provide a method for genomic analysis. The method includes using a computer system comprising a processor coupled to a memory subsystem for presenting to a user a plurality of genomic tools organized into a pipeline (wherein at least a first one of the tools comprises an executable and a wrapper script), receiving instructions from the user and sequence data, wherein the instructions call for the sequence data to be analyzed by the pipeline, and selecting—using the wrapper script—a change to the pipeline. In some embodiments, the change to the pipeline comprises execution of an alternative executable instead of the executable. Optionally, the wrapper script further performs the change to the pipeline.

In certain embodiments, the wrapper script selects the change in response to an error produced by the first one of the tools. The wrapper script may recommend the change to the user and allows the user to accept the recommendation. The executable may include a sequence alignment program and the change to the pipeline may include an alternative sequence alignment program. In certain embodiments the selected change includes a request for additional resources and the wrapper script makes the request (e.g., retrieving a data file not provided by the user and not included in the sequence data; retrieving data from a URL; retrieving a matrix of probabilities; calling for a first tool in the pipeline to generate ancillary data from the sequence data to be used by a subsequent tool in the pipeline when the subsequent tool analyzes the sequence data; requesting additional computing power; requesting additional computer processors; requesting one or more virtual machines; or requesting additional storage space).

In some embodiments, the wrapper script selects the change to the pipeline by receiving an error from the executable, making an adjustment that avoids the error, and re-running the

executable. In certain embodiments, the instructions include at least one flag that establishes a value for a parameter, and the smart wrapper selects the change by changing the flag to establish a different value for the parameter. The wrapper script may select a change that comprises not analyzing the sequence data with the executable. The wrapper script may detect an inconsistency, e.g., between the instructions and the executable, between the instructions and the sequence data, or between the sequence data and the executable. Selecting the change may include recommending that the user use a second tool instead of the first one of the tools. In some embodiments, the wrapper script adds a flag to the instructions that sends a parameter to the executable, wherein the parameter controls how the executable analyzes the sequence data. The wrapper script may cause the system to: prompt the user for additional data, prompt the user to accept the selected change, inform the user of the selected change, or combinations thereof. In some embodiments, the wrapper script analyzes the sequence data and selects the change based on a feature of the sequence data.

Brief Description of the Drawings

FIG. 1 illustrates a pipeline editor.

FIG. 2 presents an overview of a workflow involving a pipeline.

FIG. 3 diagrams a system according to certain embodiments.

FIG. 4 depicts a tool that includes a wrapper script.

FIG. 5 gives a display presented by pipeline editor.

FIG. 6 illustrates a wrapper of a tool.

FIG. 7 shows a graphical representation of using a smart wrapper.

FIG. 8 illustrates how a tool may be brought into pipeline editor.

FIG. 9 illustrates functional components of a system of the invention.

FIG. 10 illustrates the operation of systems of the invention.

FIG. 11 illustrates a pipeline that converts a SAM file into a FASTQ file.

FIG. 12 shows a pipeline for differential expression analysis.

FIG. 13 shows a pipeline for providing an alignment summary.

FIG. 14 depicts a pipeline for split read alignment.

Detailed Description

FIG. 1 illustrates a pipeline editor 101 according to some embodiments. Pipeline editor 101 may be presented in any suitable format such as a dedicated computer application or as a web site accessible via a web browser. Generally, pipeline editor 101 will present a work area in which a user can see and access icons representing a plurality of tools 107a, 107b,...,107n. Tools may be dragged from sidebar 801 into the workspace of editor 101 and connected to one another by connectors 501. Any tool 107n may include a wrapper script 233n and a binary executable 401n. In certain embodiments, executable 401n will be a sequence analysis executable. Wrapper script 233 evaluates and reacts to parameters or inputs given to tool 107, any input data, the associated executable 401n, the environment in which tool 107 is running, or errors generated by executable 401n. A novel feature of the invention is that a wrapper script 233 can identify, suggest, or implement a change to pipeline 113. A change may be, to illustrate, running an alternative executable 401m instead of executable 401n as caused by wrapper script 233n.

Tool 107 may be represented within pipeline editor 101 as an icon. In general, a tool 107 will have at least one input or output that can be linked to one or more input or output of another tool 107. The inputs and outputs of the tools can be represented graphically as little symbols (nodules) attached to the icon. A set of linked tools may be referred to as a pipeline. The graphical user interface of pipeline editor 101 allows a user to link pairs of the executables via their respective output and input streams to define a pipeline.

Selecting (e.g., clicking on) a tool allows parameters of that tool to be set (see FIG. 5). The parameters are then passed on during execution by the wrappers (see, e.g., FIG. 10). A pipeline 113 can be built by connecting combinations of the tools with connectors 501 that represent data-flows from one tool to another. FIGS. 11-14 illustrate a variety of sample pipelines in which the files that serve as the pipeline's inputs and outputs may be represented as nodes, just like tools. Input files are connected via connectors to the input nodules on the tools they serve as inputs for, and output files are connected to the output nodules on the tools that generate them. Input and output nodes can represent single files, or they can represent multidimensional data structures such as a list of files, a list of lists of files, others, or a combination thereof.

In some embodiments, input and output files consist of sequence data and associated meta-data, including file type (.bam, .fastq, etc.) along with other properties such as sample id,

date created, author, others, or a combination thereof. Preferably, input file types and dimensions will match that required by the tool being fed. Where a tool 107 includes a sequence analysis executable, the sequence analysis executable will generally define an input stream and an output stream (represented as input and output points, respectively, of corresponding tool 107).

FIG. 2 presents an overview of a workflow involving a pipeline 113 according to a certain implementation of the invention. Pipeline module 809 is a system component that runs pipelines 113. Pipeline module 809 executes a tool 107 by running wrapper script 233 (which may be provided by scripts—such as Python scripts). Wrapper script 233 calls executable 401, sets the parameters and inputs (in accord with either what the user has selected, what previous tools in the pipeline have generated, what the execution environment requires, or sensible defaults), sets the output file paths, runs executable 401 and passes along any errors thrown.

Wrapper script 233 does more than just run tool executable 401 and return the tool's outputs or errors. Wrapper script 233 can suggest that pipeline module 809 do something other than what is strictly indicated by the design of pipeline 113, the input data, or the user's instructions to get a desired result.

In some embodiments, pipeline module 809 will follow the suggestions from wrapper script 233 automatically by default, but if the wrapper script 233 includes a "prompt" job, then pipeline module 809 will instead pass along the suggestion to the user for a decision on whether or not to follow the suggestion (this is important in cases where the suggestion from wrapper script 233 may alter the results obtained). In some cases, the wrapper script 233 may include a "notify" job instead, which would signal to pipeline module 809 to go ahead and follow the suggestion but send a heads up message to the user informing them of the change.

Wrapper script 233 can log or record the suggestions and any changes made to the optimized pipeline 237 run as a result of those suggestions or changes from wrapper script 233, to ensure reproducibility, allow for debugging, inform users, and other such functionality. Wrapper script 233 can perform a variety of functions including such broad categories of functions as proposing an alternative job, requesting additional resources, and recovering from errors intelligently.

One important category of functions provided by a wrapper script 233 includes proposing an alternative job. A wrapper script 233 can evaluate the parameters and inputs it has been given

and suggest to pipeline module 809 that a different set of parameters and inputs or even running a different tool would be better for getting the desired result (see FIG. 10).

Instead of returning outputs or an error, wrapper script 233 essentially returns "run THIS instead", where THIS fully describes the alternate job including tools, parameters, and inputs.

Reasons why wrapper script 233 might propose an alternative job include: (i) some combination of input data, tools 107, and user instructions and parameters will result in an error; (ii) an alternate set of input data, tools 107, and user instructions and parameters might run more efficiently, saving the user time or money (e.g., where the user pays for execution costs); (iii) the parameters and inputs given strongly suggest a user error, and therefore running the job as ordered would be a waste (this would call for the "notify" job); and (iv) an alternate set of input data, tools 107, and user instructions and parameters will give a 'better' result from a scientific standpoint (e.g., a more accurate alignment) without significant tradeoffs (this would be a good place for the "prompt" job, since the user should make the ultimate call on substantive scientific questions).

The alternative job proposed by wrapper script 233 can actually be a set of jobs. For example, wrapper script 233 may suggest that the system "run this (some other) pipeline", or "run this tool and then take its outputs and feed it into this next tool", or "run these tools (or several instances of the same tool) in parallel".

One important category of functions provided by a wrapper script 233 includes requesting additional resources. A wrapper script 233 can also evaluate the resources a tool 107 has available to it on the machine (e.g., Amazon EC2 instance) that the tool 107 is running on, and tell pipeline module 809 that tool 107 needs additional resources to do the job. Resources requested might include elements of the execution environment, such as extra computing power or memory. Resources requested might also include particular files/data, specified by URL, which are then saved in a cache to ensure reproducibility even if the version at the URL changes.

Just as proposing an alternative job can include proposing an alternative set of jobs, requesting an additionally resource can be a multi-step process. For example, wrapper script 233 may issue an instruction that says, in essence, "go to the database at URL X, enter this SQL query, and provide me with the output."

One important category of functions provided by a wrapper script 233 includes recovering from errors intelligently. While some of the wrapper script 233 functions described

here take place before the tool is run, wrapper script 233 can also evaluate errors thrown by a tool and suggest an alternative that would avoid the error. The suggested alternatives can take the form of different parameters/tools/inputs or additional resources.

In some embodiments, pipeline module 809 requests AWS Elastic Cloud Compute (EC2) instances (e.g., to provide command module 819 in FIG. 10) for running tools from tool module 813, the component which abstracts EC2 service and keeps a "pool" of available instances. Pipeline module 809 decides what sort of instance is needed based on wrapper metadata, which contains information on the resources (CPU, memory, storage) a tool requires, sometimes including specific resource requests for particular sub-jobs. In the depicted implementation, pipeline module 809 causes a tool module 813 to execute individual tools 401. User input (e.g., in the form of sequence files) is run through pipeline 113, with wrapper 233 reading inputs, instructions, metadata, and executables and controlling the flow of sequence data through pipeline 113. Since a wrapper 233 can actually cause substantive changes to pipeline 113 (e.g., cause executable 401b to run instead of 401a), it can be thought of that wrapper 233 provides an organized optimized pipeline 237, which provides the output.

Generally, a smart wrapper 233 is included in a tool 107 along with a sequence analysis executable 401. When a pipeline 113 calls tool 107n, the wrapper script 233n of that tool 107n calls executable 401n. Sequence analysis executables can include, for example, GATK, Paup*, MrBayes, etc. Any such executable 401n may be a compiled, executable binary (e.g., accessible at /bin). The corresponding wrapper script 233n generally includes a command to execute executable 401n and may include information to manage input or output data, settings flags, error codes, logging, running a program in the background, or other such functionality that will be appreciated by one of skill in the art. A wrapper script may be created in any suitable language known in the art including, for example, bash, Perl, Python, or others. FIG. 2 illustrates that a smart wrapper 233 can be understood as contributing an optimized pipeline 237 from a pipeline 113.

As discussed above, a pipeline generally refers to a bioinformatics workflow that includes one or a plurality of individual steps. Each step (embodied and represented as a tool 107 within pipeline editor 101) generally includes an analysis or process to be performed on genetic data. For example, an analytical project may begin by obtaining a plurality of sequence reads. The pipeline editor 101 can provide the tools to quality control the reads and then to assemble

the reads into contigs. The contigs may then be compared to a references, such as the human genome (e.g., hg18) to detect mutations by a third tool. These three tools—quality control, assembly, and compare to reference—as used on the raw sequence reads represent but one of myriad genomic pipelines. Genomic pipelines are discussed in Dinov, 2011, Applications of the pipeline environment for visual informatics and genomic computations, BMC Bioinf 12:304 and Torri, 2012, Next generation sequence analysis and computational genomics using graphical pipeline workflows, Genes (Basel) 3:545.

As represented in FIG. 1, each step is provided as a tool 107. Any tool 107 may perform any suitable analysis such as, for example, alignment, variant calling, RNA splice modeling, quality control, data processing (e.g., of FASTQ, BAM/SAM, or VCF files), or other formatting or conversion utilities. Pipeline editor 101 represents tools 107 as "apps" and allows a user to assemble tools into a pipeline 113.

Small pipelines can be included that use but a single app, or tool. For example, editor 101 can include a merge FASTQ pipeline that can be re-used in any context to merge FASTQ files. Complex pipelines that include multiple interactions among multiple tools (e.g., such as a pipeline to call variants from single samples using BWA+ GATK) can be created to store and reproduce published analyses so that later researchers can replicate the analyses on their own data. Using the pipeline editor 101, a user can browse stored tools and pipelines to find a stored tool 107 of interest that offers desired functionality. The user can then copy the tool 107 of interest into a project, then run it as-is or modify it to suit the project. Additionally, the user can build new analyses from scratch.

Embodiments of the invention can include server computer systems that provide pipeline editor 101 as well as computing resources for performing the analyses represented by pipeline 113. Computing execution and storage can be provided by one or more server computers of the system, by an affiliated cloud resource, by a user's local computer resources, or a combination thereof.

FIG. 3 diagrams a system 201 according to certain embodiments. System 201 generally includes a server computer system 207 to provide functionality such as access to one or more tools 107. A user can access pipeline editor 101 and tools 107 through the use of a local computer 213. A pipeline module on server 207 can invoke the series of tools 107 called by a

Date Reçue/Date Received 2022-02-04

pipeline 113. A tool module can then invoke the commands or program code called by the tool 107. Commands or program code can be executed by processing resources of server 207. In certain embodiments, processing is provided by an affiliated cloud computing resource 219. Additionally, affiliated storage 223 may be used to store data.

A user can interaction with pipeline editor 101 through a local computer 213. Local computer 213 can be a laptop, desktop, or mobile device such as a tablet or smartphone. In general, local computer 213 is a computer device that includes a memory coupled to a processor with one or more input/output mechanism. Local computer 213 communicates with server 207, which is generally a computer that includes a memory coupled to a processor with one or more input/output mechanism. These computing devices can optionally communicate with affiliated resource 219 or affiliated storage 223, each of which preferably use and include at least computer comprising a memory coupled to a processor.

As one skilled in the art would recognize as necessary or best-suited for performance of the methods of the invention, systems of the invention include one or more computer devices that include one or more processors (e.g., a central processing unit (CPU), a graphics processing unit (GPU), etc.), computer-readable storage devices (e.g., main memory, static memory, etc.), or combinations thereof which communicate with each other via a bus. A computer generally includes at least one processor coupled to a memory via a bus and input or output devices.

A processor may be any suitable processor known in the art, such as the processor sold under the trademark XEON E7 by Intel (Santa Clara, CA) or the processor sold under the trademark OPTERON 6200 by AMD (Sunnyvale, CA).

Memory preferably includes at least one tangible, non-transitory medium capable of storing: one or more sets of instructions executable to cause the system to perform functions described herein (e.g., software embodying any methodology or function found herein); data (e.g., embodying any tangible physical objects such as the genetic sequences found in a patient's chromosomes); or both. While the computer-readable storage device can in an exemplary embodiment be a single medium, the term "computer-readable storage device" should be taken to include a single medium or multiple media (e.g., a centralized or distributed database, and/or associated caches and servers) that store the instructions or data. The term "computer-readable storage device" shall accordingly be taken to include, without limit, solid-state memories (e.g.,

subscriber identity module (SIM) card, secure digital card (SD card), micro SD card, or solid-state drive (SSD)), optical and magnetic media, and any other tangible storage media.

Any suitable services can be used for affiliated resource 219 or affiliated storage 223 such as, for example, Amazon Web Services. In some embodiments, affiliated storage 223 is provided by Amazon Elastic Block Store (Amazon EBS) snapshots, allowing cloud resource 219 to dynamically mount Amazon EBS volumes with the data needed to run pipeline 113. Use of cloud storage 223 allows researchers to analyze data sets that are massive or data sets in which the size of the data set varies greatly and unpredictably. Thus, systems of the invention can be used to analyze, for example, hundreds of whole human genomes at once.

Input/output devices according to the invention may include a video display unit (e.g., a liquid crystal display (LCD) or a cathode ray tube (CRT) monitor), an alphanumeric input device (e.g., a keyboard), a cursor control device (e.g., a mouse or trackpad), a disk drive unit, a signal generation device (e.g., a speaker), a touchscreen, an accelerometer, a microphone, a cellular radio frequency antenna, and a network interface device, which can be, for example, a network interface card (NIC), Wi-Fi card, or cellular modem.

As shown in FIG. 1, within pipeline editor 101, individual tools (e.g., command line tools) are represented as an icon in a graphical editor.

FIG. 4 depicts a tool 107, shown represented as an icon 301. Tool 107 includes wrapper script 233, which has the ability to call executable 401. Icon 301 may have one or more output point 307 and one or more input point 315 corresponding to output and input pipes, respectively, of executable 401. In embodiments in which a tool 107 includes an underlying sequence analysis executable, input point 315 is analogous to an argument or data that can be piped in and output point 307 represents the output of the command. Icon 301 may be displayed with a label 311 to aid in recognizing tool 107. In some embodiments, selecting, or single-clicking on, the icon 301 for tool 107 allows parameters of the tool to be set within pipeline editor 101.

When a pipeline 113 that includes tool 107 is run, at the point during the pipeline workflow where tool 107 is to be called, pipeline module 809 will call wrapper script 233. In the illustrative embodiment shown in FIG. 4, script 233 is a Python script that checks first to see if the variable ref has been assigned the contents of file hg18 (here shown in a simplified pseudo-code for illustrative purposes). If hg18 has not been assigned to ref, script 233 exits and tells the user that a reference is required. In the illustrated example, executable 401 is Mosaik aligner,

which aligns reads to a reference. A user has set up wrapper script 233 to require hg18 as the reference that Mosaik will use. The user has thus used wrapper script 233 to interrupt the running of pipeline 113 in the event, for example, that the reference is set to hg19. If ref has been set to hg18, then wrapper 233 issues the system command MosaikAligner which causes executable 401 to run. Script 233 can pass along the switches or flags as well as the data to Mosaik. This described functionality is accessible via pipeline editor 101.

FIG. 5 gives a display presented by pipeline editor 101 when a tool 107 is selected. The tool may include buttons for deleting that tool or getting more information associated with the icon 301. Additionally, a list of parameters for running the tool may be displayed with elements such as tick-boxes or input prompts for setting the parameters (e.g., analogous to switches or flags in UNIX/LINUX commands). Clicking on tool 107 allows parameters of the tool to be set within editor 101 (e.g., within a GI). As discussed in more detail below, the parameter settings will then be passed through the tool module to the command-level module. A user may build pipeline 113 by placing connectors between input points 315 and output points 307.

FIG. 6 illustrates how a wrapper 233b sits beneath a tool 107b within a pipeline 113. Here, pipeline 133 includes a connector 501 connecting a first tool 107a to a second tool 107b. Connector 501 represents a data-flow from first tool 107a to second tool 107b (e.g., analogous to the pipe (|) character in UNIX/LINUX text commands). Wrapper 233b evaluates the output of tool 107a, instructions and flags (i.e., switches or parameters) from a user, an executable associated with tool 107b, and can respond to any inconsistency among those. For example, the command "bamtools merge" may be invoked by wrapper 233b to call bamtools merge as executable 401b. Wrapper 233b expects the output of tool 107a to thus be numerous small BAM files. In a given instance, a user may be running a job that will cause tool 107a to output only a single BAM file. In this instance, wrapper 233b may detect that inconsistence between the input to tool 107b and the corresponding executable 401b, and may be pre-programmed to, under those facts, simply skip tool 107b without further comment (or optionally to give a notification).

FIG. 7 shows a graphical representation of using a smart wrapper 233 to keep an analysis running even where there is an inconsistency between a user's instructions and the input data. Here, pipeline 713 includes Mosaik as tool 107a, and a user has set up pipeline to align hg18 to hg19. Wrapper script 233a detects that the user's instructions to align hg18 to hg19 are not consistent with the use of Mosaik, which expects to align numerous short reads to a reference.

Script 233a identifies that pipeline 713 can be changed to include MUMmer instead of Mosaik. This can be accomplished by any suitable means. For example, script 233a can include a table or a series of "if...elseif..." statements that assign input to specific aligners based on qualities of the input. The qualities of the input that script 233a examines include, for example, file size, extension, file format, number of input files, metadata, or other information. In the illustrate example, script 233a may recognize that a set of files with a *.vcf extension and one genome-sized file are suitable for Mosaik. However, script 233a may recognize that two files of substantially equal size are not suited to being aligned by Mosaik and are suited to be aligned by MUMmer. See, e.g., Delcher, et al., 1999, Alignment of whole genomes, Nucleic Acids Research 27(11):2369-2376. Script 233a identifies that pipeline 713 should be updated so that it would execute MUMmer as executable 401b. In some embodiments, script 233a will simply make that change, and MUMmer will align hg18 to hg19. It is worth noting that the updated pipeline 713 will call MUMmer as tool 107b, and that this may call script 233b.

FIG. 8 illustrates how a tool 107 may be brought into pipeline editor 101 for use within the editor. In some embodiments, pipeline editor 101 includes an "apps list" 801 shown in FIG. 8 as a column to the left of the workspace in which available tools are listed. In some embodiments, apps on apps list 801 can be dragged out into the workspace where they will appear as icons. A user can perform a drag gesture to bring any tool (i.e., any App) into the workspace of pipeline editor 101.

Systems described herein may be embodied in a client/server architecture. Alternatively, functionality described herein may be provided by a computer program application that runs solely on a client computer (i.e., runs locally). A client computer can be a laptop or desktop computer, a portable device such as a tablet or smartphone, or specialized computing hardware such as is associated with a sequencing instrument. For example, in some embodiments, functions described herein are provided by an analytical unit of an NGS sequencing system, accessing a database according to embodiments of the invention and assembling sequence reads from NGS and reporting results through the terminal hardware (e.g., monitor, keyboard, and mouse) connected directly to the NGS system. In some embodiments, this functionality is provided as a "plug-in" or functional component of sequence assembly and reporting software such as, for example, the GS De Novo Assembler, known as gsAssembler or Newbler (NEW assemBLER) from 454 Life Sciences, a Roche Company (Branford, CT). Newbler is designed to

assemble reads from sequencing systems such as the GS FLX+ from 454 Life Sciences (described, e.g., in Kumar, S. et al., Genomics 11:571 (2010) and Margulies, et al., Nature 437:376-380 (2005)). In some embodiments, a production application is provided as functionality within a sequence analyzing system such as the HiSeq 2500/1500 system or the Genome AnalyzerIIX system sold by Illumina, Inc. (San Diego, CA) (for example, as downloadable content, an upgrade, or a software component).

FIG. 9 illustrates functional components of a system 201 according to certain embodiments. Generally, a user will interact with a user interface (UI) 801 provided within, for example, local computer 213. A UI module 805 may operate within server system 207 to send instructions to and receive input from UI 801. Within server system 207, UI module 805 sits on top of pipeline module 809 which executes pipelines 113. Pipeline module 809 executes wrapper scripts 233. Pipeline module 809 directly handles scheduling and execution of tasks, while an independent component may be employed to allocated instances and make sure they're being used efficiently. The running, or execution, of tools 107 is done by the wrapper scripts 233 (see FIG. 10 for more detail).

Preferably, UI module 801, pipeline module 809, and tool module 813 are provided at least in part by server system 207. In some embodiments, affiliated cloud computing resource 219 contributes the functionality of one or more of UI module 801, pipeline module 809, and tool module 813. Command-level module 819 may be provided by one or more of local computer 213, server system 207, cloud computing resource 219, or a combination thereof. It is noted that as drawn in FIG. 10, the ">" character does not represent the info line prefix of a FASTA file but instead here represents a UNIX prompt to show that command module 819 hypothetically receives the commands for tools p, q, r, x, y, and z to be executed with output piped to input along the chain.

Computer program instructions can be written using any suitable language known in the art including, for example, Perl, BioPerl, Python, C++, C#, JavaScript, Ruby on Rails, Groovy and Grails, or others. Program code can be linear, object-oriented, or a combination thereof. Preferably, program instructions for the tools described here are provided as distinct modules, each with a defined functionality. Exemplary languages, systems, and development environments include Perl, C++, Python, Ruby on Rails, JAVA, Groovy, Grails, Visual Basic .NET. An overview of resources useful in the invention is presented in Barnes (Ed.),

Bioinformatics for Geneticists: A Bioinformatics Primer for the Analysis of Genetic Data, Wiley, Chichester, West Sussex, England (2007) and Dudley and Butte, A quick guide for developing effective bioinformatics programming skills, PLoS Comput Biol 5(12):e1000589 (2009).

In some embodiments, systems of the invention are developed in Perl (e.g., optionally using BioPerl). Perl is discussed in Tisdall, Mastering Perl for Bioinformatics, O'Reilly & Associates, Inc., Sebastopol, CA 2003. In some embodiments, tools 107 are developed using BioPerl, a collection of Perl modules that allows for object-oriented development of bioinformatics applications. BioPerl is available for download from the website of the Comprehensive Perl Archive Network (CPAN). See also Dwyer, Genomic Perl, Cambridge University Press (2003) and Zak, CGI/Perl, 1st Edition, Thomson Learning (2002).

In certain embodiments, systems of the invention are developed using Java and optionally the BioJava collection of objects, developed at EBI/Sanger in 1998 by Matthew Pocock and Thomas Down. BioJava provides an application programming interface (API) and is discussed in Holland, et al., BioJava: an open-source framework for bioinformatics, Bioinformatics 24(18):2096-2097 (2008). Java is discussed in Liang, Introduction to Java Programming, Comprehensive (8th Edition), Prentice Hall, Upper Saddle River, NJ (2011) and in Poo, et al., Object-Oriented Programming and Java, Springer Singapore, Singapore, 322 p. (2008).

Systems of the invention can be developed using the Ruby programming language and optionally BioRuby, Ruby on Rails, or a combination thereof. Ruby or BioRuby can be implemented in Linux, Mac OS X, and Windows as well as, with JRuby, on the Java Virtual Machine, and supports object oriented development. See Metz, Practical Object-Oriented Design in Ruby: An Agile Primer, Addison-Wesley (2012) and Goto, et al., BioRuby: bioinformatics software for the Ruby programming language, Bioinformatics 26(20):2617-2619 (2010).

Systems and methods of the invention can be developed using the Groovy programming language and the web development framework Grails. Grails is an open source model-view-controller (MVC) web framework and development platform that provides domain classes that carry application data for display by the view. Grails domain classes can generate the underlying database schema. Grails provides a development platform for applications including web applications, as well as a database and an object relational mapping framework called Grails Object Relational Mapping (GORM). The GORM can map objects to relational databases and represent relationships between those objects. GORM relies on the Hibernate object-relational

persistence framework to map complex domain classes to relational database tables. Grails further includes the Jetty web container and server and a web page layout framework (SiteMesh) to create web components. Groovy and Grails are discussed in Judd, et al., Beginning Groovy and Grails, Apress, Berkeley, CA, 414 p. (2008); Brown, The Definitive Guide to Grails, Apress, Berkeley, CA, 618 p. (2009).

FIG. 10 illustrates the operation and inter-relation of components of systems of the invention. In certain embodiments, a pipeline 113 is stored within pipeline module 809. Pipeline 113 may be represented using any suitable language or format known in the art. In some embodiments, a pipeline is described and stored using JavaScript Object Notation (JSON). The pipeline JSON objects include a section describing nodes (nodes include tools 107 as well as input points 315 and output points 307) and a section describing the relations (i.e., connections 501) between the nodes.

Pipeline module 809 actually executes wrapper scripts 233 and may also be the component that executes these pipelines 113. Running or executing the wrapper scripts 233 is what runs or executes the tools 107.

Tool module 813 manages information about the wrapped tools 107 that make up pipelines 113 (such as inputs/outputs and resource requirements). Tool module 813 stores the wrappers 233. The executables 401 may themselves comprise one or any number of commands (e.g., l, m, n,... or p, q, r,...or x, y, z..., to illustrate).

The UI module 805 handles the front-end user interface. This module can represent workflows from pipeline module 809 graphically as pipelines in the graphical pipeline editor 101. The UI module can also represent the tools 107 that make up the nodes in each pipeline 113 as node icons 301 in the graphical editor 101, generating input points 315 and output points 307 and tool parameters from the information in tool module 813. The UI module will list other tools 107 in the "Apps" list along the side of the editor 101, from whence the tools 107 can be dragged and dropped into the pipeline editing space as node icons 301.

In certain embodiments, UI module 805, in addition to listing tools 107 in the "Apps" list, will also list other pipelines the user has access to (separated into "Public Pipelines" and "Your Custom Pipelines"), getting this information from pipeline module 809.

Using systems described herein, a wide variety of genomic analytical pipelines may be provided. In general, pipelines will relate to analyzing genetic sequence data. The variety of

pipelines that can be created is open-ended and unlimited. In some embodiments, one or more pipelines may be included in system 201 as a tool for use in pipeline editor 101. For example, certain genomic analytical steps may be routine and common and thus conducive to be being offered as a pre-made pipeline.

To illustrate the breadth of possible analyses that can be supported using system 201 and to introduce a few exemplary pipelines that may be included for use within a system of the invention, a few example pipelines are discussed.

FIG. 11 illustrates a relatively simple pipeline 1001 that converts a sequence alignment map (SAM) file or a binary version of a SAM (BAM) into a FASTQ file.

FIG. 12 shows a pipeline 1101 for differential expression analysis using the program Cuffdiff. Pipeline 1101 can find significant differences in transcript expression between groups of samples. In pipeline 1101, Cuffdiff accepts read alignment files from any number of groups containing one or more samples, it calculates expression levels at the isoform and gene level, and it tests for significant expression differences. Cuffdiff outputs a downloadable collection of files, viewable as spreadsheets that can be explored. This pipeline can also perform basic quality control of differential expression experiment powered by CummeRbund. Lastly, pipeline 1101 can render interactive visualizations from Cuffdiff results. This allows a user to explore differential expression results in the form of interactive plots, export gene sets, and generate publication quality figures.

Another analysis included in a system of the invention can provide an alignment summary.

FIG. 13 shows a pipeline 1201 for providing an alignment summary. Pipeline 1201 can be used to analyze the quality of read alignment for both genomic and transcriptomic experiments. Pipeline 1201 gives useful statistics to help judge the quality of an alignment. Pipeline 1201 takes aligned reads in BAM format and a reference FASTA to which they were aligned as input, and provides a report with information such as the proportion of reads that could not be aligned and the percentage of reads that passed quality checks.

FIG. 14 depicts a pipeline 1301 for split read alignment. Pipeline 1301 uses the TopHat aligner to map sequence reads to a reference transcriptome and identify novel splice junctions. The TopHat aligner is discussed in Trapnell, et al., TopHat: discovering splice junctions with RNA-Seq. Bioinformatics 2009, 25:1105-1111. Pipeline 1301

Date Reçue/Date Received 2022-02-04

accommodates the most common experimental designs. The TopHat tool is highly versatile and the pipeline editor 101 allows a researcher to build pipelines to exploit its many functions.

Other possible pipelines can be created or included with systems of the invention. For example, a pipeline can be provided for exome variant calling using BWA and GATK.

An exome variant calling pipeline using BWA and GATK can be used for analyzing data from exome sequencing experiments. It replicates the default bioinformatics pipeline used by the Broad Institute and the 1000 Genomes Project. GATK is discussed in McKenna, et al., 2010, The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data, Genome Res. 20:1297-303 and in DePristo, et al., 2011, A framework for variation discovery and genotyping using next-generation DNA sequencing data, Nature Genetics. 43:491-498. The exome variant calling pipeline can be used to align sequence read files to a reference genome and identify single nucleotide polymorphisms (SNPs) and short insertions and deletions (indels).

Other pipelines that can be included in systems of the invention illustrate the range and versatility of genomic analysis that can be performed using system 201. System 201 can include pipelines that: assesse the quality of raw sequencing reads using the FastQC tool; align FASTQ sequencing read files to a reference genome and identify single nucleotide polymorphisms (SNPs); assess the quality of exome sequencing library preparation and also optionally calculate and visualize coverage statistics; analyze exome sequencing data produced by Ion Torrent sequencing machines; merge multiple FASTQ files into a single FASTQ file; read from FASTQ files generated by the Ion Proton, based on the two step alignment method for Ion Proton transcriptome data; other; or any combination of any tool or pipeline discussed herein.

The invention provides systems and methods for creating tools and integrating tools into a pipeline editor. Any suitable method of creating and integrating tools can be used. In some embodiments, a software development kit (SDK) is provided. In certain embodiments, a system of the invention includes a Python SDK. An SDK may be optimized to provide straightforward wrapping, testing, and integration of tools into scalable Apps. The system may include a map-reduce-like framework to allow for parallel processing integration of tools that do not support parallelization natively.

Apps can either be released across the platform or deployed privately for a user group to deploy within their tasks. Custom pipelines can be kept private within a chosen user group.

Date Reçue/Date Received 2022-02-04

Systems of the invention can include tools for security and privacy. System 201 can be used to treat data as private and the property of a user or affiliated group. The system can be configured so that even system administrators cannot access data without permission of the owner. In certain embodiments, the security of pipeline editor 101 is provided by a comprehensive encryption and authentication framework, including HTTPS-only web access, SSL-only data transfer, Signed URL data access, Services authentication, TrueCrypt support, and SSL-only services access.

Additionally, systems of the invention can be provided to include reference data. Any suitable genomic data may be stored for use within the system. Examples include: the latest builds of the human genome and other popular model organisms; up-to-date reference SNPs from dbSNP; gold standard indels from the 1000 Genomes Project and the Broad Institute; exome capture kit annotations from Illumina, Agilent, Nimblegen, and Ion Torrent; transcript annotations; small test data for experimenting with pipelines (e.g., for new users).

In some embodiments, reference data is made available within the context of a database included in the system. Any suitable database structure may be used including relational databases, object-oriented databases, and others. In some embodiments, reference data is stored in a relational database such as a "not-only SQL" (NoSQL) database. In certain embodiments, a graph database is included within systems of the invention.

Using a relational database such as a NoSQL database allows real world information to be modeled with fidelity and allows complexity to be represented.

A graph database such as, for example, Neo4j, can be included to build upon a graph model. Labeled nodes (for informational entities) are connected via directed, typed relationships. Both nodes and relationships may hold arbitrary properties (key-value pairs). There need not be any rigid schema, and node-labels and relationship-types can encode any amount and type of meta-data. Graphs can be imported into and exported out of a graph data base and the relationships depicted in the graph can be treated as records in the database. This allows nodes and the connections between them to be navigated and referenced in real time (i.e., where some prior art many-JOIN SQL-queries in a relational database are associated with an exponential slowdown).

Date Reçue/Date Received 2022-02-04

Equivalents

Various modifications of the invention and many further embodiments thereof, in addition to those shown and described herein, will become apparent to those skilled in the art from the full contents of this document, including references to the scientific and patent literature cited herein. The subject matter herein contains important information, exemplification and guidance that can be adapted to the practice of this invention in its various embodiments and equivalents thereof.

Date Reçue/Date Received 2022-02-04

What is claimed is:

1. A system for genomic analysis, the system comprising:

a processor coupled to a memory operable to cause the system to:

present to a user a plurality of genomic tools organized into a bioinformatics pipeline, wherein at least a first one of the genomic tools comprises an executable and a wrapper script;

receive instructions from the user and sequence data, wherein the instructions call for the sequence data to be analyzed by the bioinformatics pipeline;

initiate the executable of the first genomic tool of the bioinformatics pipeline; and

modify, using the wrapper script, the bioinformatics pipeline to replace the first genomic tool of the bioinformatics pipeline with an alternative genomic tool, wherein the wrapper script modifies the bioinformatics pipeline by:

receiving an error from the executable of the first genomic tool;

identifying the alternative genomic tool that is consistent with the user instructions and the sequence data being passed; and

initiating an executable of the alternative genomic tool, wherein the alternative genomic tool avoids the error.

2. The system of claim 1, wherein the wrapper script selects modifies the bioinformatics pipeline in response to the received error.

3. The system of claim 1, wherein the wrapper script recommends the modification to the user and allows the user to accept the recommendation.

4. The system of claim 1, wherein the executable comprises a sequence alignment program and the modification to the bioinformatics pipeline includes an alternative sequence alignment program.

5. The system of claim 1, wherein the selected modification includes a request for additional resources and the wrapper script makes the request.

Blacklined Copy

Date Reçue/Date Received 2022-06-27

6. The system of claim 5, wherein the request for additional resources comprises one selected from the list consisting of:

retrieving a data file not provided by the user and not included in the sequence data; retrieving data from a URL;

retrieving a matrix of probabilities;

calling for a first tool in the bioinformatics pipeline to generate ancillary data from the sequence data to be used by a subsequent tool in the bioinformatics pipeline when the subsequent tool analyzes the sequence data;

requesting additional computing power;

requesting additional computer processors;

requesting one or more virtual machines; and

requesting additional storage space.

7. The system of claim 1, wherein the instructions include at least one flag that establishes a value for a parameter, and the wrapper script selects the modification by changing the flag to establish a different value for the parameter.

8. The system of claim 1, wherein the wrapper script selects a modification that comprises not analyzing the sequence data with the executable.

9. The system of claim 1, wherein the wrapper script detects an inconsistency between the instructions and the executable.

10. The system of claim 1, wherein the wrapper script detects an inconsistency between the instructions and the sequence data.

11. The system of claim 1, wherein the wrapper script detects an inconsistency between the sequence data and the executable.

12. The system of claim 1, wherein modifying the bioinformatics pipeline comprises recommending that the user use the alternative genomic tool instead of the first genomic tool.

Date Reçue/Date Received 2022-06-27

13. The system of claim 1, wherein the wrapper script adds a flag to the instructions that sends a parameter to the executable, wherein the parameter controls how the executable analyzes the sequence data.

14. The system of claim 1, wherein the wrapper script causes the system to prompt the user for additional data.

15. The system of claim 1, wherein the wrapper script causes the system to prompt the user to accept the selected change.

16. The system of claim 1, wherein the wrapper script causes the system to inform the user of the selected change.

17. The system of claim 1, wherein the wrapper script analyzes the sequence data and selects the change based on a feature of the sequence data.

18. The system of claim 4, wherein the wrapper script includes a series of statements that assign input data to specific sequence alignment programs based on qualities of the input data.

19. The system of claim 18, wherein the qualities are at least one of the following: file size, extension, file format, number of input files, and metadata.

20. A method for processing a bioinformatics pipeline, the method comprising:

receiving, from a user, instructions to process a bioinformatics pipeline, the bioinformatics pipeline comprising a plurality of genomic tools, wherein at least a first one of the genomic tools comprises an executable and wrapper metadata;

creating a first job for execution, the first job comprising the executable of the first one of the genomic tools and input data, wherein the first job further comprises a cloud instance;

modifying the bioinformatics pipeline to avoid an error relating to the executable of the first genomic tool, the modification comprising replacing the executable of the first job with an executable of an alternative genomic tool according to the wrapper metadata of the first one of the genomic tools; and

initiating execution of the first job on the cloud instance, wherein the modification of the first job avoids the error.

21. The method of claim 20, further comprising modifying the bioinformatics pipeline to avoid an error that relates to an insufficient resource condition.

22. The method of claim 21, wherein modifying the bioinformatics pipeline to avoid an error related to an insufficient resource condition comprises determining a need for additional resources from the wrapper metadata, and requesting the additional resources for execution of the alternative genomic tool.

23. The method of claim 22, wherein the requested additional resources include sufficient computing power to avoid the insufficient resource condition.

24. The method of claim 23, wherein the requested additional resources include sufficient computer processors to avoid the insufficient resource condition.

25. The method of claim 22, wherein the requested additional resources include sufficient storage space to avoid the insufficient resource condition.

26. The method of claim 20, wherein creating the first job for execution further comprises:

initiating execution, on a first cloud instance, of the first job; and

receiving, from the first cloud instance, the error from the executable of the first genomic tool;

wherein modifying the bioinformatics pipeline is performed in response to receiving the error.

27. The method of claim 20, wherein the executable includes a sequence alignment program and the alternative genomic tool includes an alternative sequence alignment program.

28. The method of claim 20, wherein replacing the executable of the first genomic tool further comprises replacing the first job with a set of jobs.

Date Reçue/Date Received 2022-06-27

29. The method of claim 28, wherein replacing the first job with a set of jobs comprises calling for a second one of the tools in the bioinformatics pipeline to generate ancillary data from the input data, the ancillary data to be used as input data by the executable of the alternative genomic tool.

30. The method of claim 22, wherein the requested additional resources include a data file not provided by the user and not included in the input data.

31. The method of claim 20, wherein modifying the bioinformatics pipeline to avoid an error further comprises adding a flag to instructions that send a parameter to the executable of the alternative genomic tool, wherein the parameter controls how the executable of the alternative genomic tool analyzes the input data.

32. The method of claim 20, wherein the wrapper metadata comprises a script that detects an inconsistency.

33. The method of claim 32, wherein the wrapper script detects an inconsistency between the executable and input data of the first job.

34. The method of claim 32, wherein the wrapper script modifies the bioinformatics pipeline to avoid the error.

35. The method of claim 20, further comprising prompting the user to allow the modification.

36. The method of claim 20, wherein the cloud instance is selected based on the wrapper metadata.

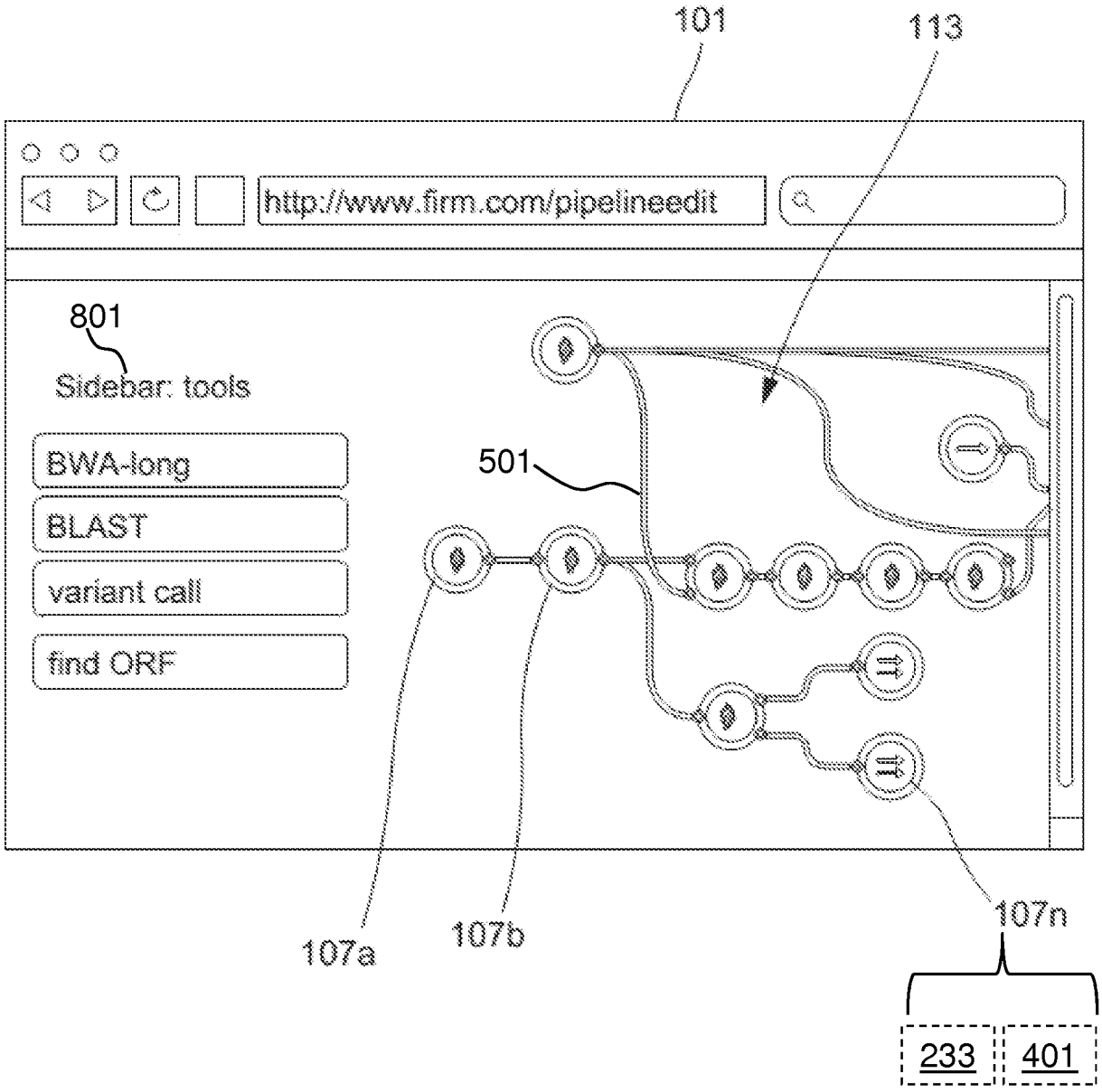

101
113
http://www.firm.com/pipelineedit
801
Sidebar: tools
BWA-long
BLAST
variant call
find ORF
501
107a
107b
107n
233
401

FIG. 1

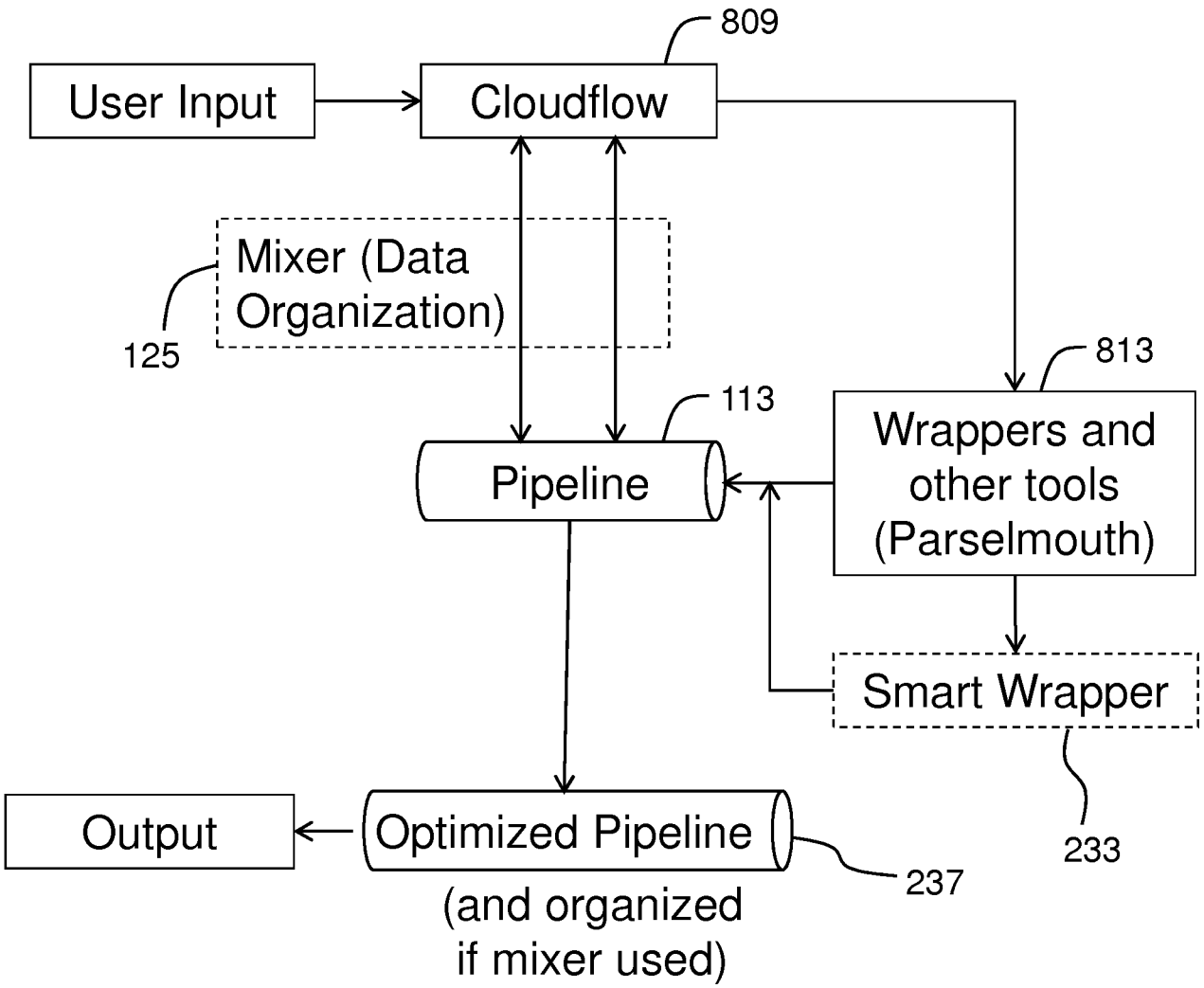

809
User Input
Cloudflow
Mixer (Data Organization)
125
813
113
Wrappers and other tools (Parselmouth)
Pipeline
Smart Wrapper
233
Output
Optimized Pipeline
237
(and organized if mixer used)

FIG. 2

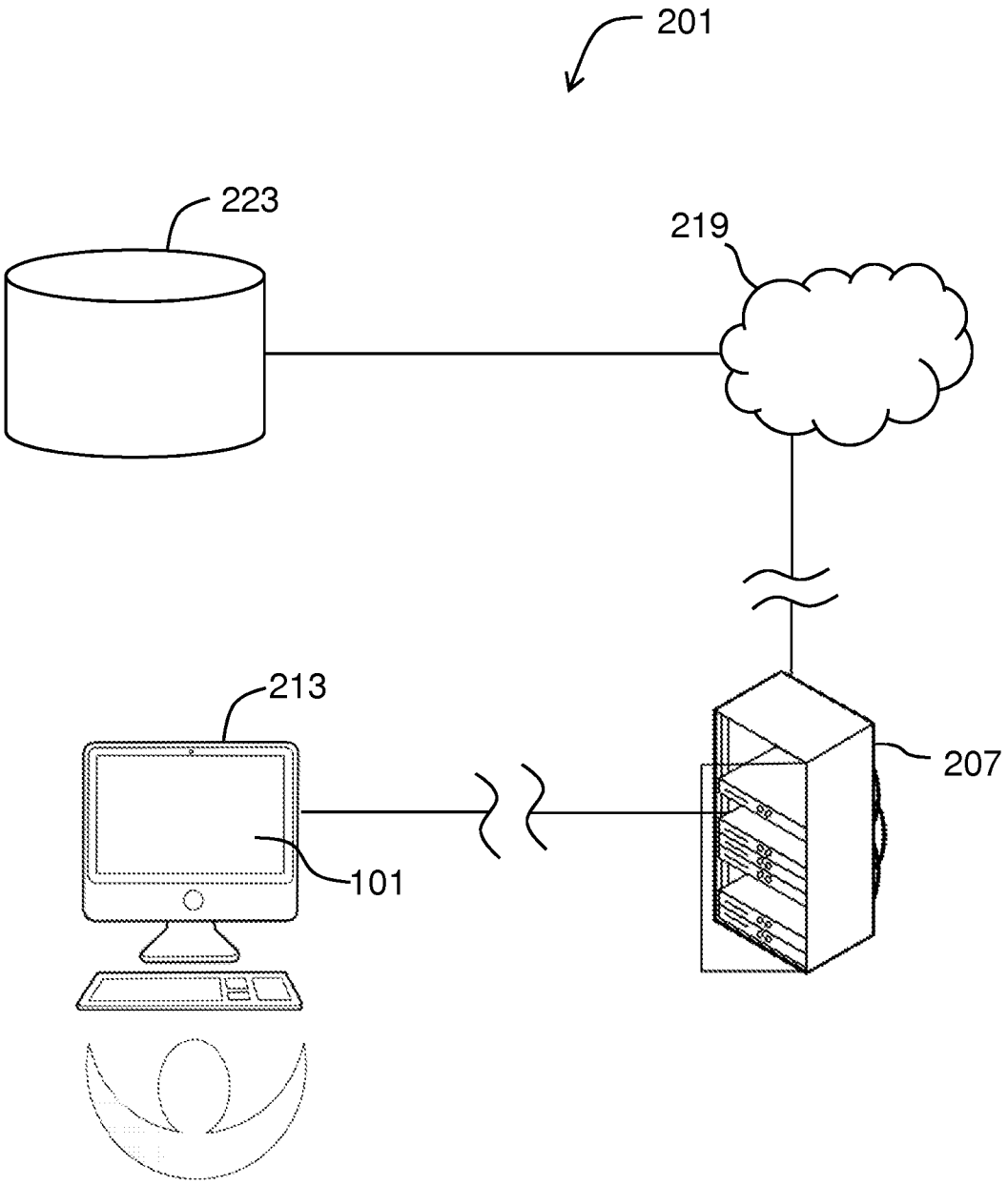


FIG. 3

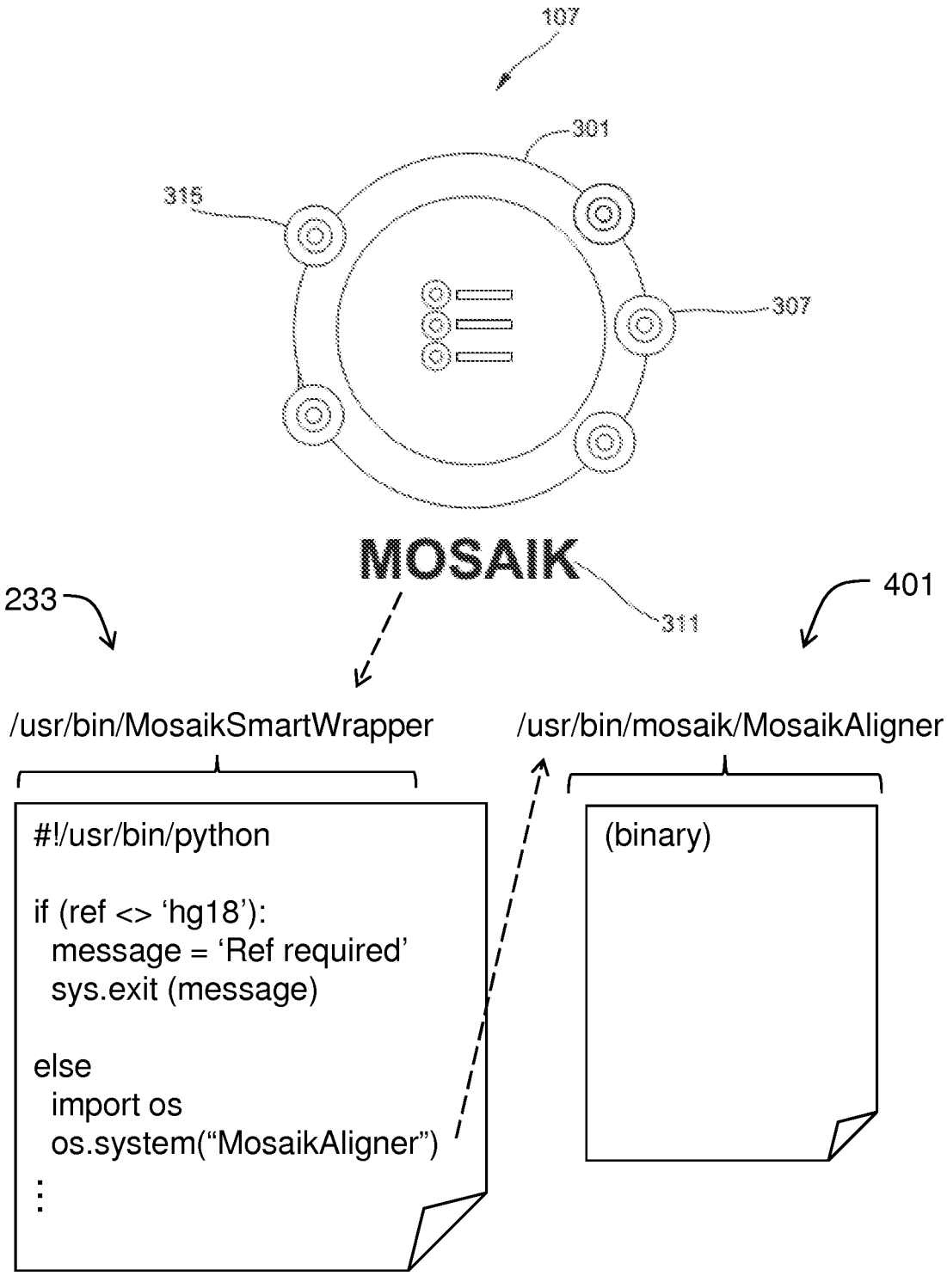

107
301
315
307
MOSAIK
311
233
401
/usr/bin/MosaikSmartWrapper
/usr/bin/mosaik/MosaikAligner
#!/usr/bin/python
if (ref <> 'hg18'):
message = 'Ref required'
sys.exit (message)
else
import os
os.system("MosaikAligner")
(binary)

FIG. 4

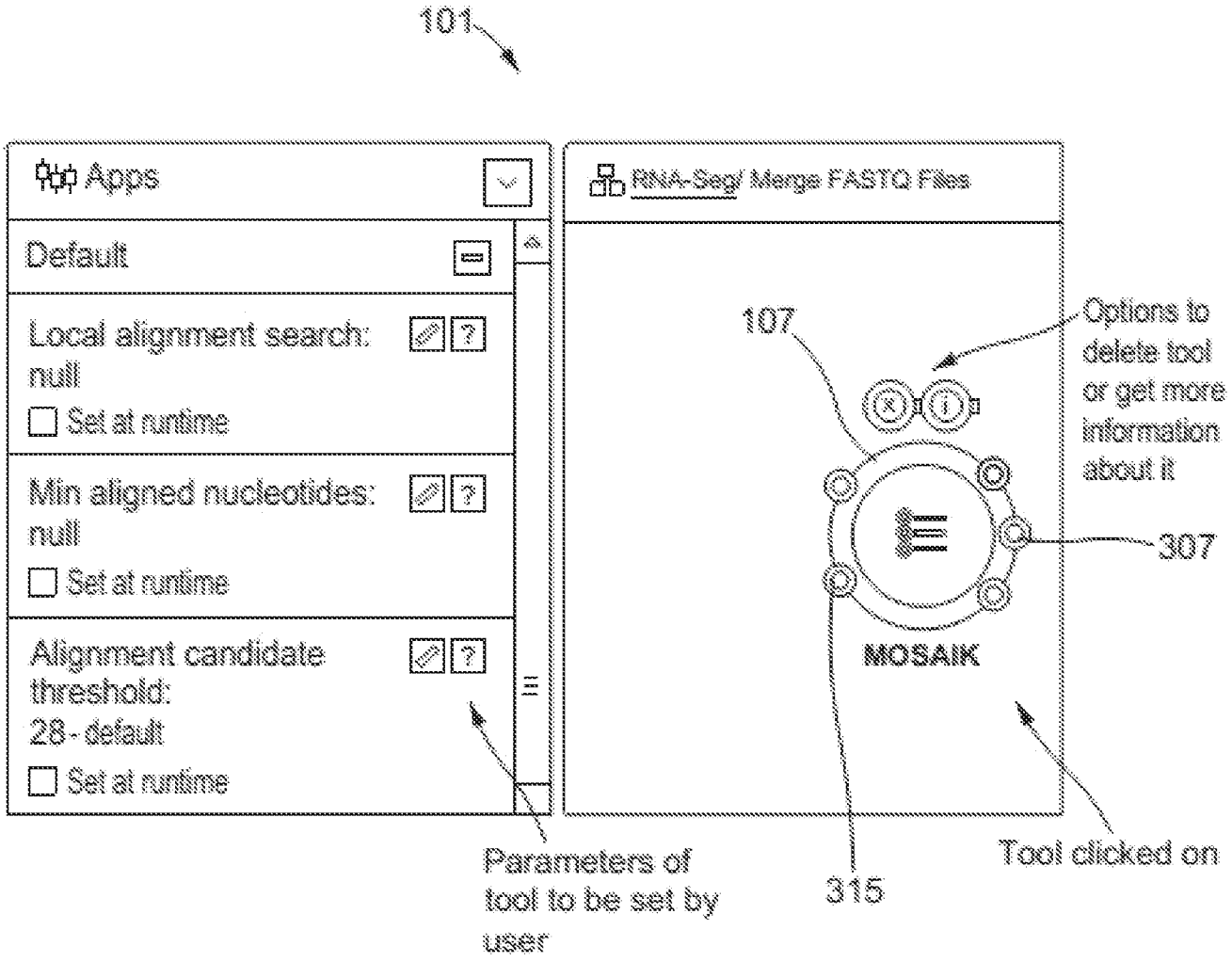

101
Apps
Default
Local alignment search:
null
Set at runtime
Min aligned nucleotides:
null
Set at runtime
Alignment candidate threshold:
28 - default
Set at runtime
RNA-Seq/ Merge FASTQ Files
107
Options to delete tool or get more information about it
307
MOSAIK
315
Parameters of tool to be set by user
Tool clicked on

FIG. 5

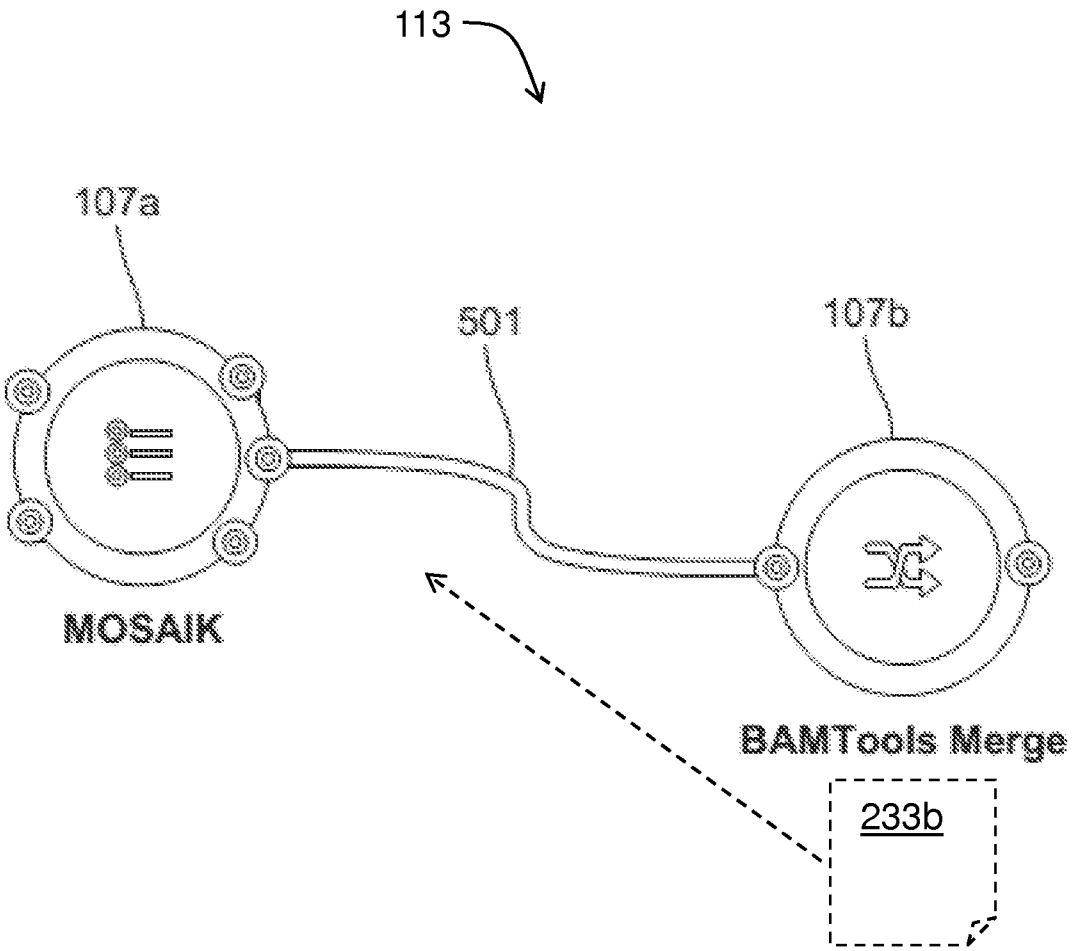


FIG. 6

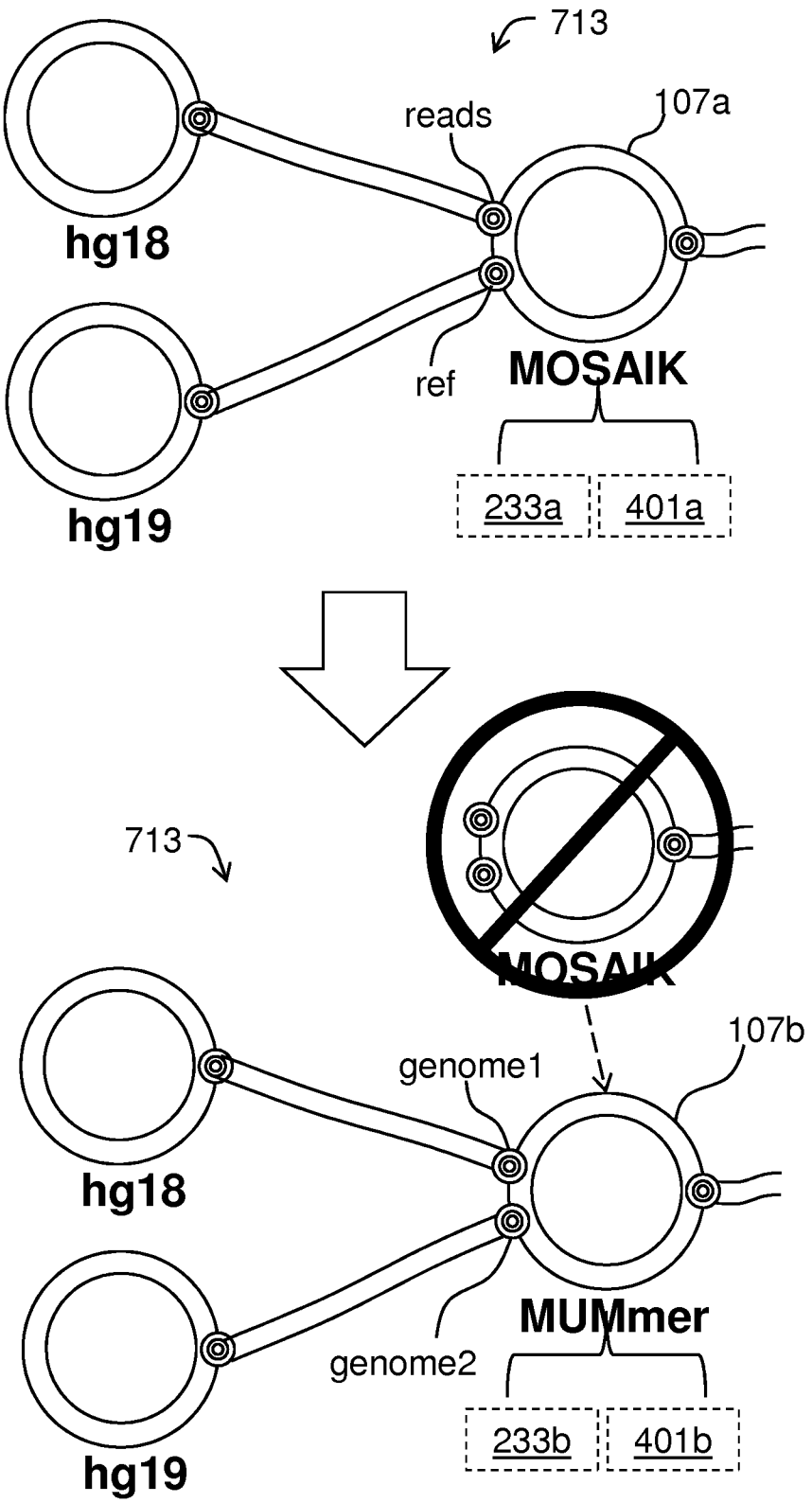

713
reads
107a
hg18
ref
MOSAIK
hg19
233a
401a
713
MOSAIK
107b
genome1
hg18
MUMmer
genome2
hg19
233b
401b

FIG. 7

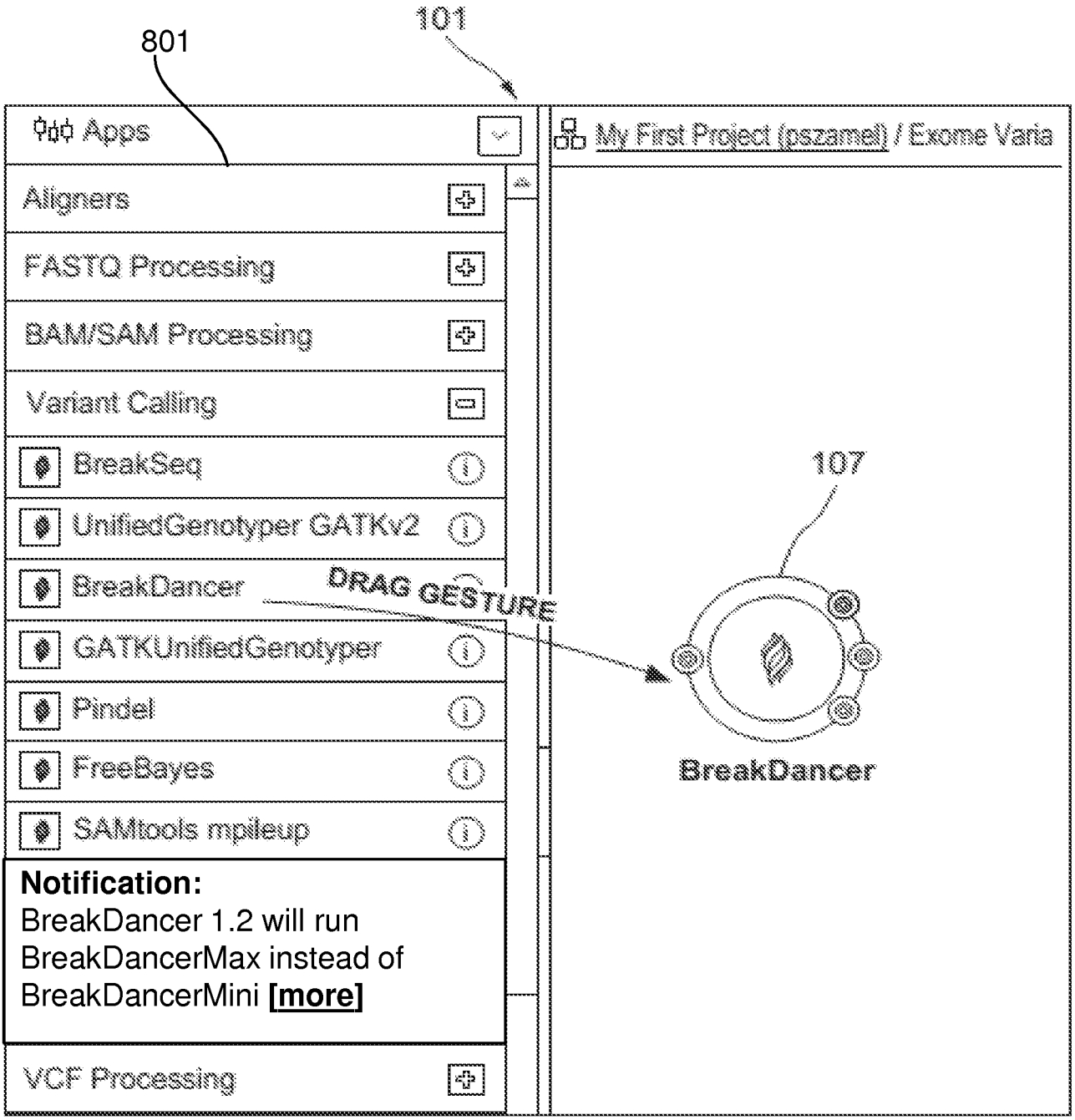

801
101
Apps
My First Project (pszamel) / Exome Varia
Aligners
FASTQ Processing
BAM/SAM Processing
Variant Calling
BreakSeq
UnifiedGenotyper GATKv2
BreakDancer
GATKUnifiedGenotyper
Pindel
FreeBayes
SAMtools mpileup
DRAG GESTURE
107
BreakDancer
Notification:
BreakDancer 1.2 will run BreakDancerMax instead of BreakDancerMini [more]
VCF Processing

FIG. 8

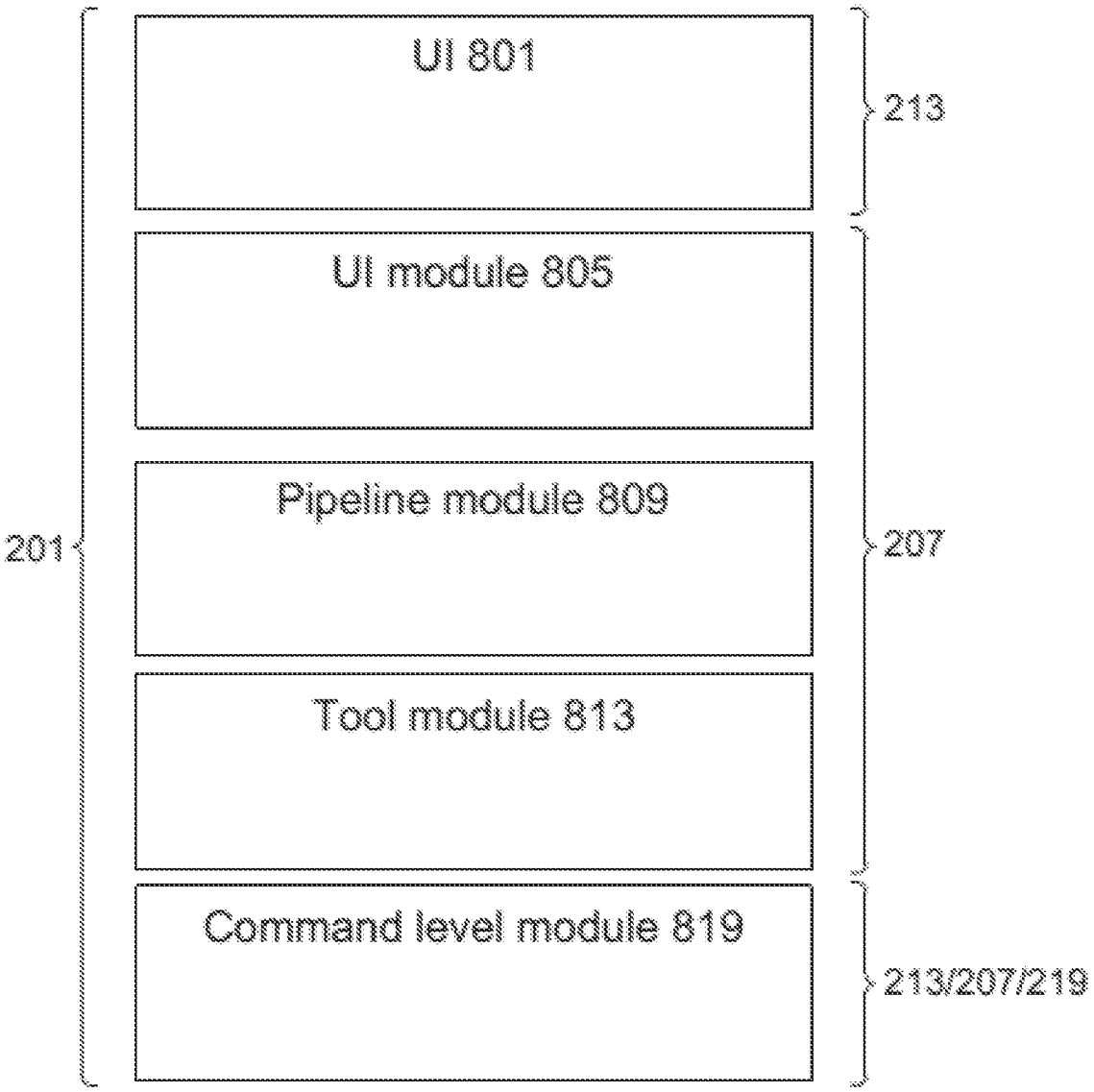

UI 801
213
UI module 805
Pipeline module 809
201
207
Tool module 813
Command level module 819
213/207/219

FIG. 9

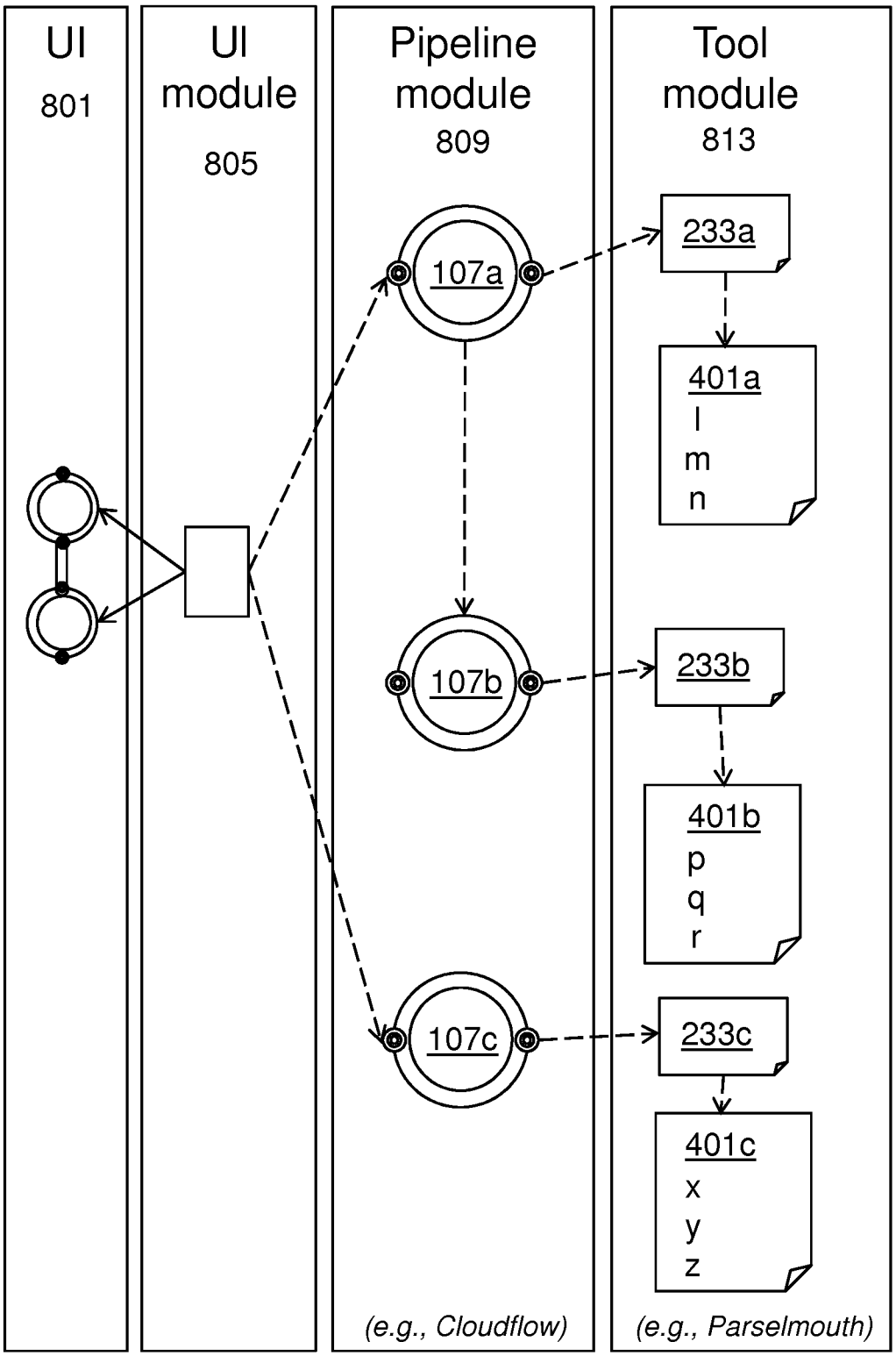

UI
801
UI
module
805
Pipeline
module
809
Tool
module
813
107a
107b
107c
233a
233b
233c
401a
l
m
n
401b
p
q
r
401c
x
y
z
(e.g., Cloudflow)
(e.g., Parselmouth)

FIG. 10

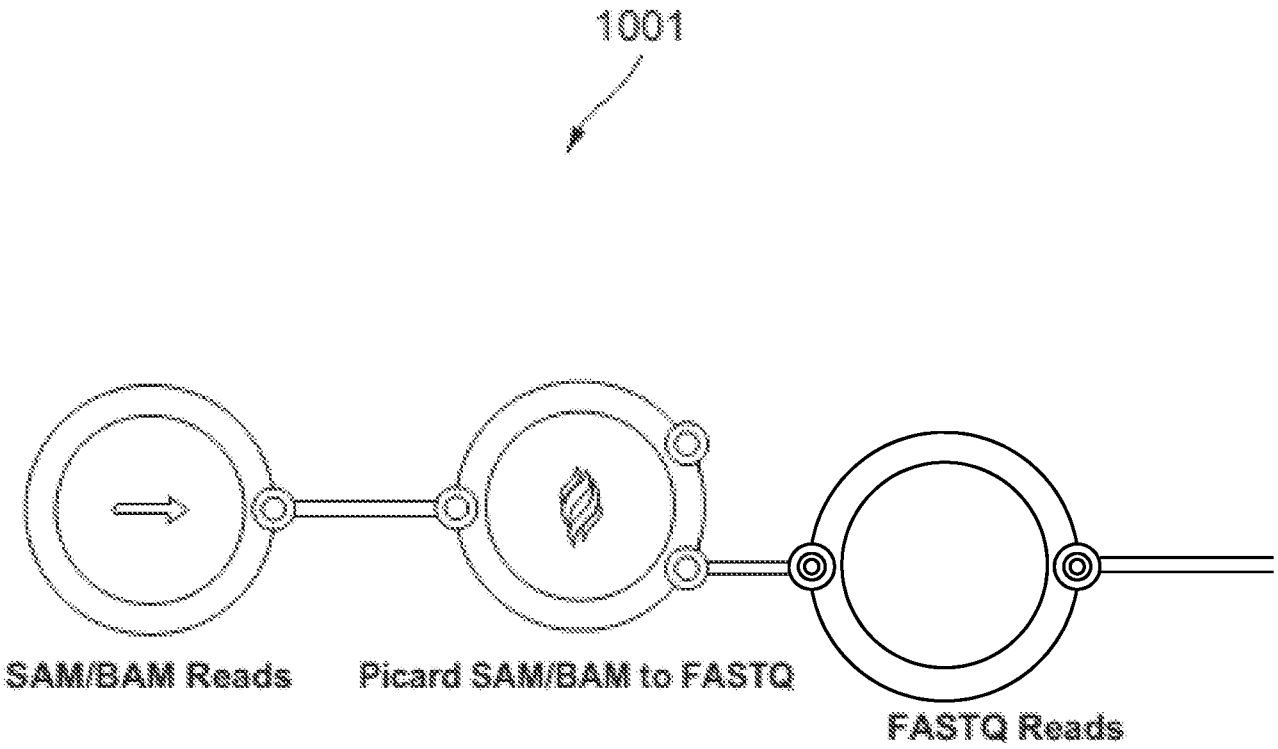


FIG. 11

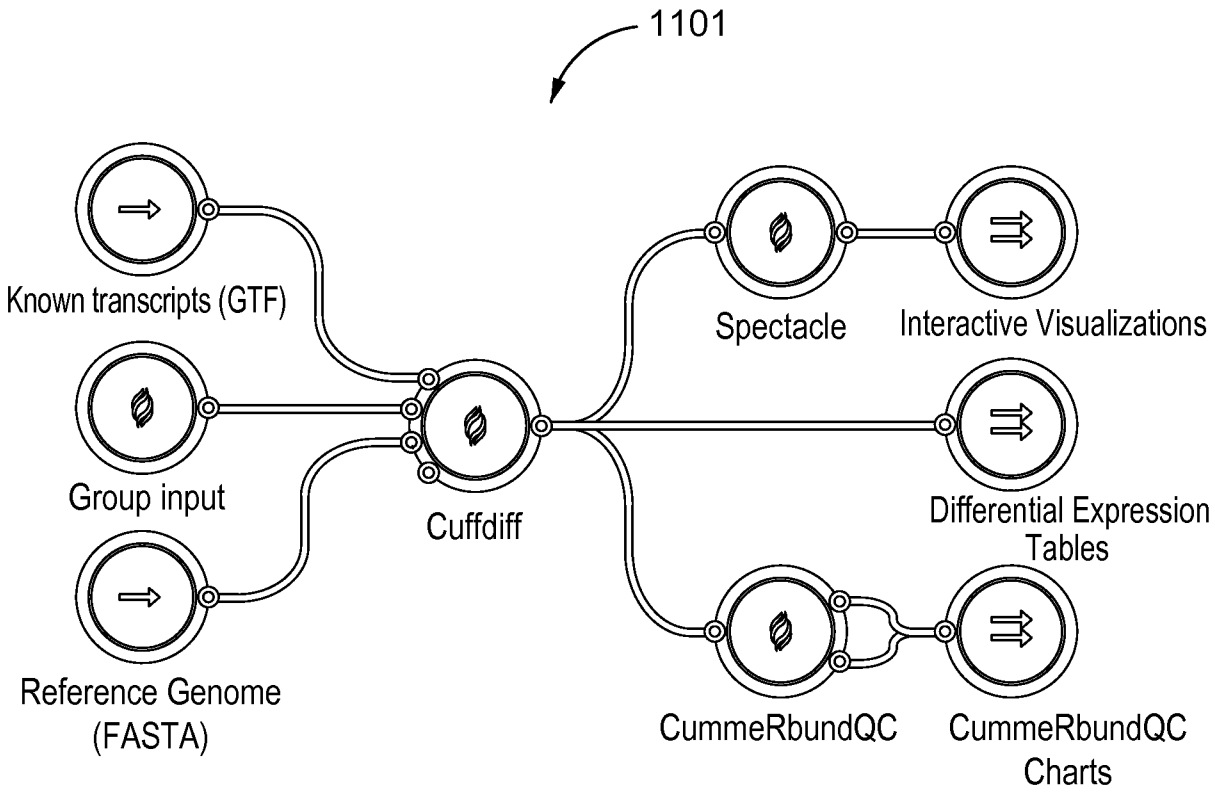

1101
Known transcripts (GTF)
Group input
Reference Genome (FASTA)
Cuffdiff
Spectacle
Interactive Visualizations
Differential Expression Tables
CummeRbundQC
CummeRbundQC Charts

FIG. 12

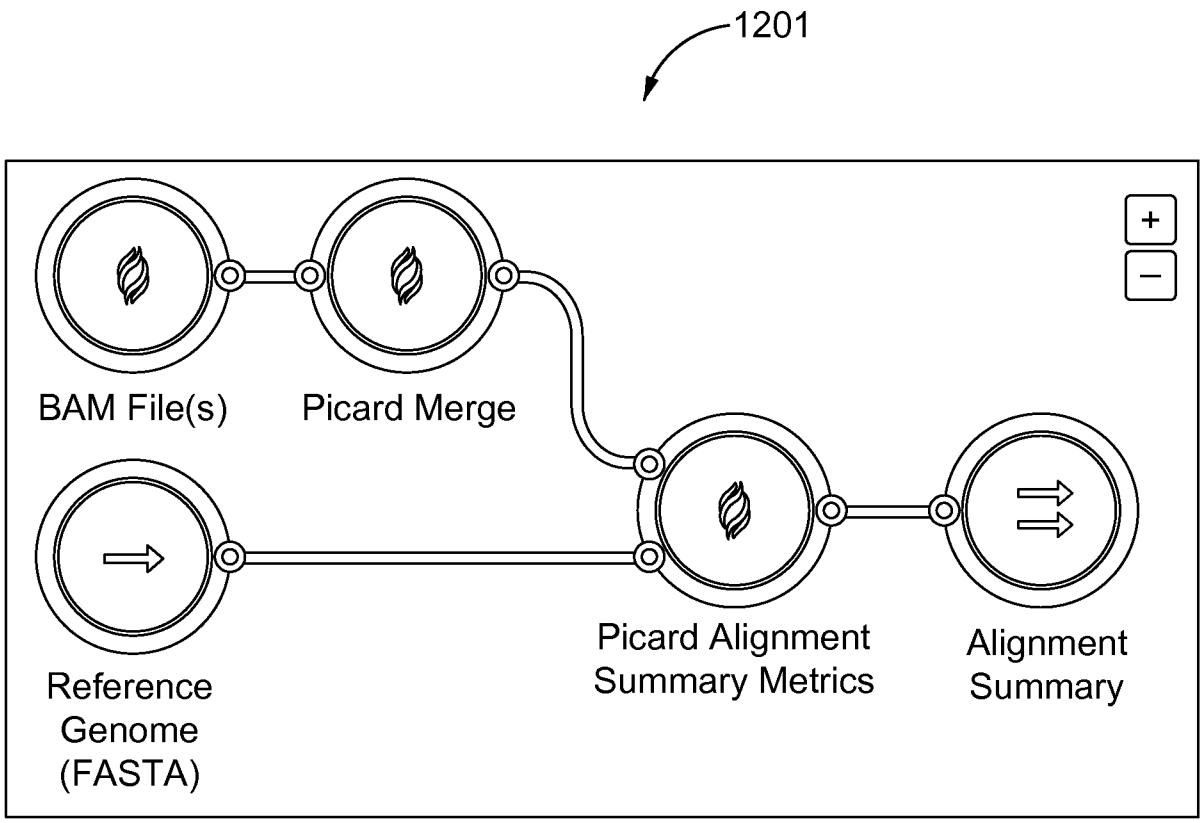

1201
BAM File(s)
Picard Merge
Reference Genome (FASTA)
Picard Alignment Summary Metrics
Alignment Summary

FIG. 13

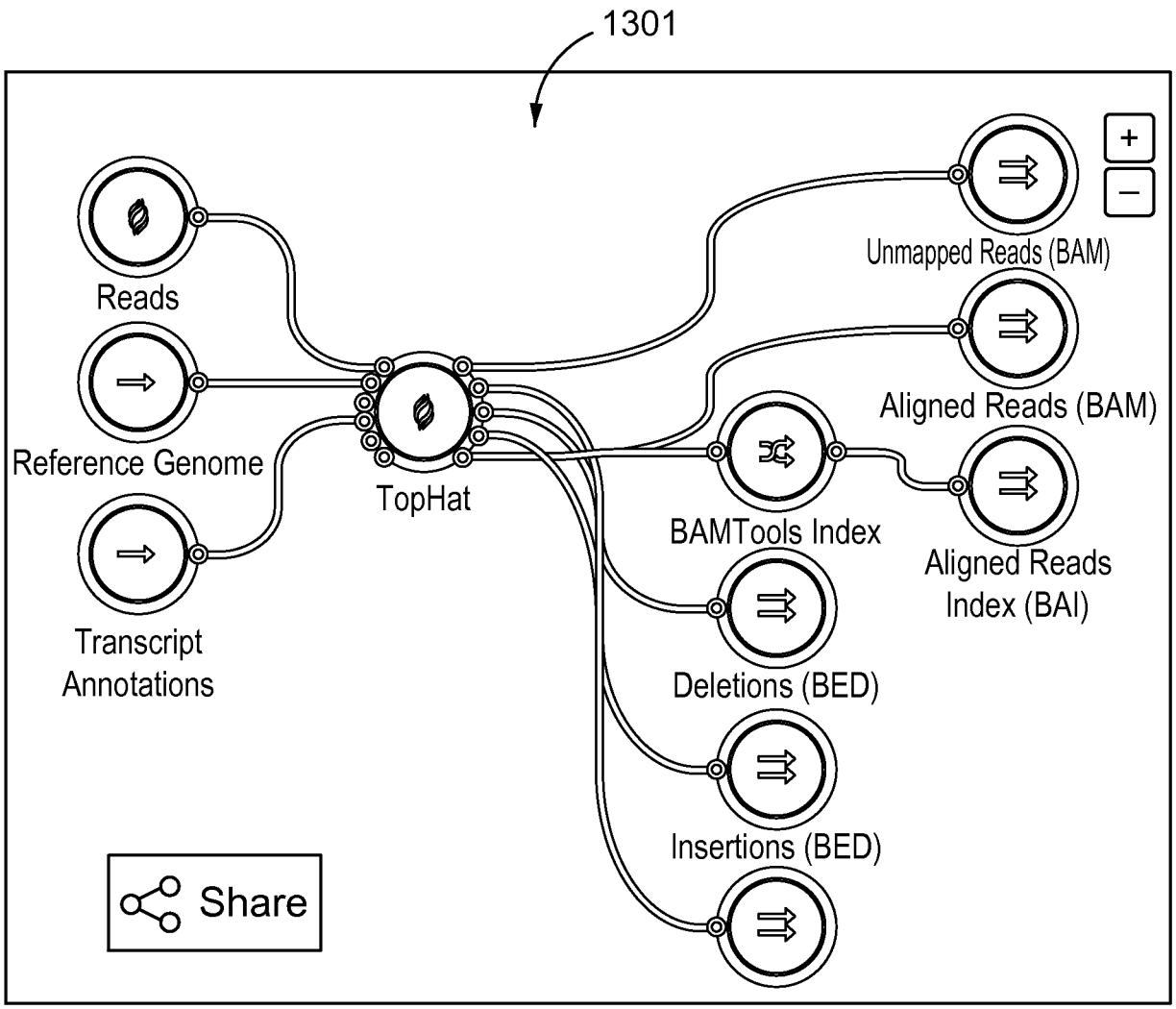
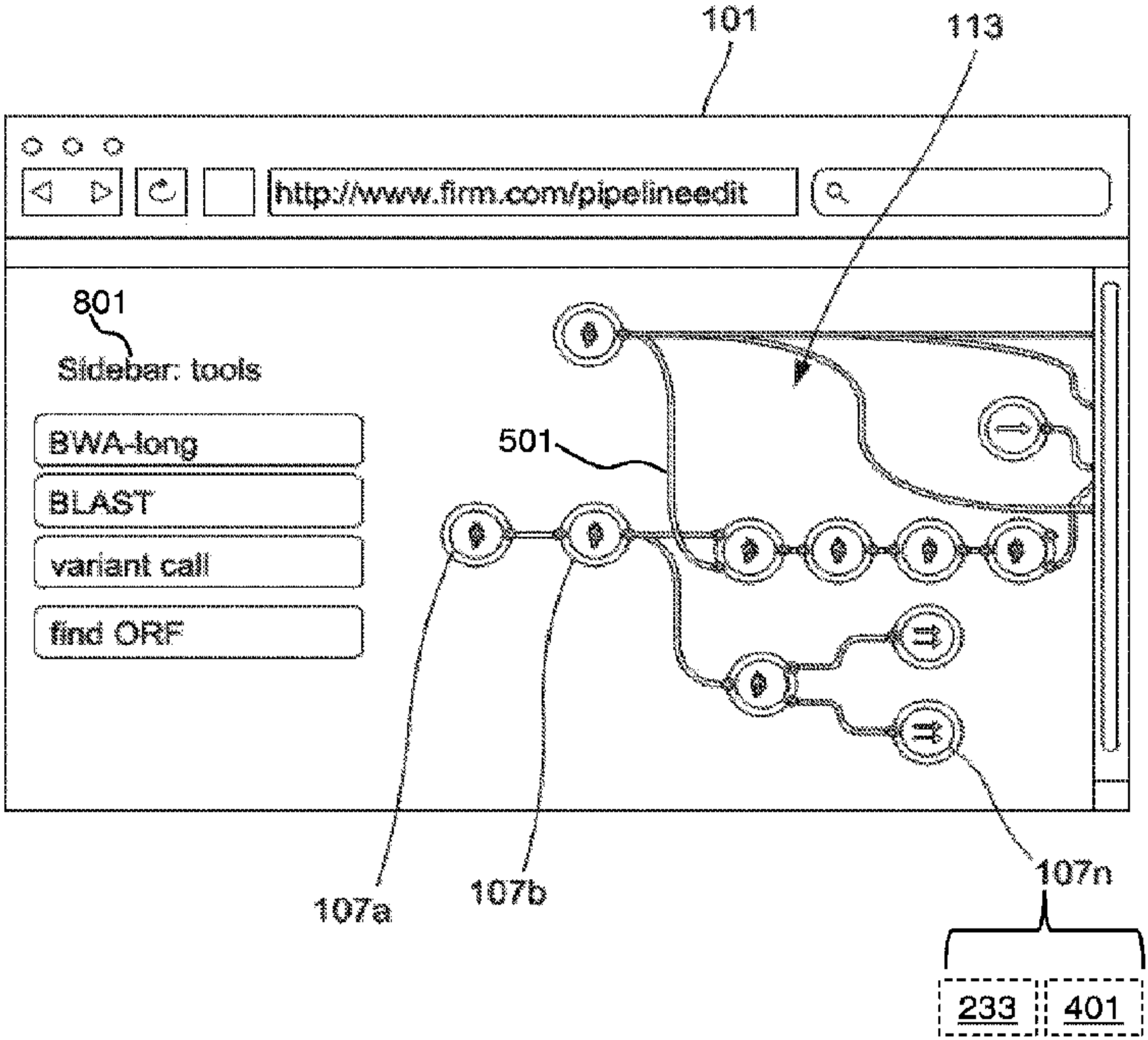

1301
Reads
Reference Genome
Transcript Annotations
TopHat
Unmapped Reads (BAM)
Aligned Reads (BAM)
BAMTools Index
Aligned Reads Index (BAI)
Deletions (BED)
Insertions (BED)
Share

FIG. 14


101
113
http://www.firm.com/pipelineedit
801
Sidebar: tools
BWA-long
BLAST
variant call
find ORF
501
107a
107b
107n
233
401