



- (51) International Patent Classification:
G06F 17/30 (2006.01)
- (21) International Application Number:
PCT/US2015/016695
- (22) International Filing Date:
19 February 2015 (19.02.2015)
- (25) Filing Language: English
- (26) Publication Language: English
- (71) Applicant: **HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP** [US/US]; 11445 Compaq Center Drive West, Houston, TX 77070 (US).
- (72) Inventors: **CHEN, Fei**; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US). **VISWANATHAN, Krishnamurthy**; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US). **JAYARAM, Nandish**; 908 Greek Row Dr., Apt. 246, Arlington, Texas 76013 (US). **GONZALEZ DIAZ, Maria T**; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US).
- (74) Agents: **KUIPERS, Luke W.** et al.; Hewlett Packard Enterprise, 3404 E. Harmony Road, Mail Stop 79, Fort Collins, CO 80528 (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,

[Continued on next page]

(54) Title: POST-SAMPLING CONFLICT DETECTION

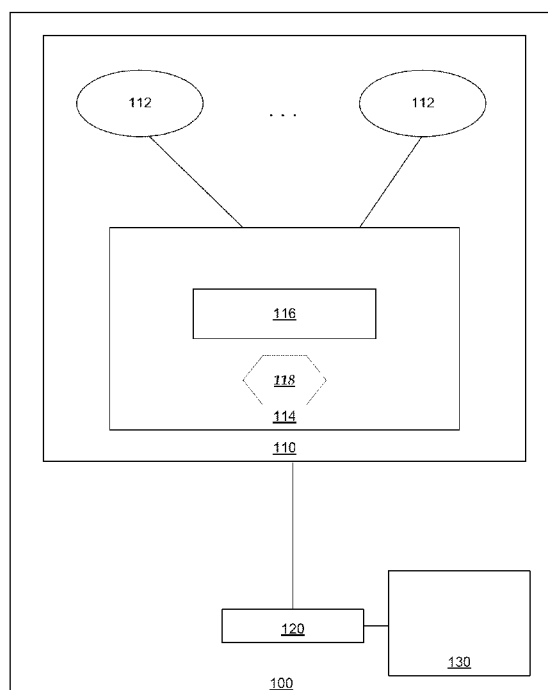


FIG. 1

(57) Abstract: Methods, devices, and techniques for detecting post-sampling conflicts in samples are discussed herein. For example, a set of random variables and dependency data specifying dependency relationships between the random variables may be received. A first random variable may then be selected from the set of random variables. Based on the dependency data, it is then identified that the first random variable depends on a second random variable. A first sample of the first random variable is then generated according to a conditional distribution. The conditional distribution may be calculated based on a value of the second random variable. After generating the first sample, a lack of a post-sampling conflict between the first sample and the second random variable is detected. Then, the first sample is committed to an output sample set.





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))*

Declarations under Rule 4.17:

— *as to the identity of the inventor (Rule 4.17(i))*

POST-SAMPLING CONFLICT DETECTION

Background

[0001] Bioinformatics, computer vision, text analytics, finance and marketing analytics, and other fields may use statistical inferences to generate predictions of expected classifications, labels, or outcomes. Such systems may, by way of example and not limitation, be configured to determine a number of people that are likely to shop at a particular store on a given day, a candidate a voter may likely vote for, and so forth.

Brief Description of the Drawings

[0002] FIG. 1 is a schematic diagram of a computer device according to an example implementation.

[0003] FIG. 2 is a flowchart illustrating a method for executing a Gibbs sampler using optimistic concurrency control, according to an example.

[0004] FIG. 3 is a flowchart illustrating a method for using timestamps to detect post-sampling conflicts, according to an example.

[0005] FIG. 4 is a flowchart that illustrates a method for detecting post-sampling conflicts in a Gibbs sampler based on comparing neighbor snapshots generated before and after the generation of a sample, according to an example.

Detailed Description

[0008] To generate prediction models, a computer system may use training data that includes a number of samples for a given feature or set of features. Using the training data, the computer system can create, for example, a histogram for a feature that can in turn be used to build or otherwise train the prediction model.

[0009] However, in some cases, it may be difficult to obtain these samples. Where it is difficult to obtain direct samples of a set of features, a computer system may execute a Gibbs sampler to approximate these samples. A Gibbs sampler may generate a sequence of samples which are approximated from a specified multivariate probability distribution (e.g., from the joint probability distribution of two or more random variables). To generate a sample for a random variable X, the Gibbs sampler may read current assignments of those variables that X depends on, compute a conditional distribution of X, sample X according to the conditional distribution, and update X's assignment with the new sample. Thus, in accordance with some examples discussed herein, the term "sample" may refer to a value generated for a random variable based on a conditional distribution for the random variable given the values of neighbor random variable (e.g., those random variables that the given random variable is dependent on). That is, in some cases, a sample may be a predicted value for a random variable given the values of the neighbor random variables. In some cases, a Gibbs sampler can be randomized in that it makes use of random numbers, and hence may produce different results each time the Gibbs sampler is run.

[0010] Approaches for parallelizing a Gibbs sampler may use read/write locks to ensure that the sampler is serializable (e.g., execution of the parallel version is equivalent to execution of a sequential version). To illustrate, assume that a selected random variable depends on values from a group of random variables, herein referred to as "neighbor random variables" to simplify discussion. In this case, the Gibbs sampler may attempt to acquire a read lock for each of the neighbor random variables before generating a sample of the selected random variable.

However, locks are an expensive system resource. Furthermore, although a value for a random variable may be independent of some subset of the set of random variables when conditioned on the neighbor values, some random variables may have many neighbor random variables (say, for example, thousands of other random variables), further increasing the use of these locks.

[0011] Examples discussed herein may detect post-sampling conflicts as a technique for providing optimistic concurrency control in a Gibbs sampler. The term “post-sample conflict” as used herein may refer to a conflict that is detected after a sample for the random variable is generated. Accordingly, a sample for a random variable can be generated without first acquiring read locks on the neighbor random variables for a selected random variable.

[0012] Some examples may detect a post-sample conflict based on comparing a change in a random variable’s dependency state data before and after generating the sample for that random variable. The term “dependency state data” as used herein may relate to data that characterizes an update to the current value to a subset of the random variables the Gibbs sampler is operating on. An example of dependency state data may include a snapshot of values for the neighbor random variables for a given time period. Another example includes a timestamp assigned to a group of random variables for a given time period. In one case, the group may be a clique. The term “clique” as used herein may refer to data that specifies a set of random variables that mutually dependent on each other.

[0013] In one aspect discussed herein, a set of random variables and dependency data (e.g., data that specifies dependency relationships between the random variables) specifying dependency relationships between the random variables may be received. A first random variable may then be selected from the set of random variables. Based on the dependency data, it is then identified that the first random variable depends on a second random variable. A first sample of the first random variable is then generated according to a conditional distribution. The conditional distribution may be calculated based on a value of the second random

- 4 -

variable. After generating the first sample, a lack of a post-sampling conflict between the first sample and the second random variable is detected. Then, the first sample is committed to an output sample set.

[0014] The figures are now discussed in greater detail. FIG. 1 is a schematic diagram of a computer device 100 according to an example implementation. The computer device 100 is a physical machine that is constructed from actual machine executable instructions, or "software," and actual hardware. The hardware of the computer device 100 includes a processing node 110, which may be part of, for example, a particular multicore central processing unit (CPU) package or multiple multicore CPU packages. As depicted in FIG. 1, the processing node 110 contains CPU processor cores 112 and a local memory 114. A CPU processor core may be a processor device suitable to read and execute processor executable instructions, such as a CPU, or an integrated circuit configured to perform a configured function.

[0015] The local memory 114 may be a storage device that stores, among other things, CPU executable instructions and data, such as optimistic Gibbs sampler module instructions 116 and an output sample set 118. The output sample set 118 may be a data that represents a number of samples generated by the CPU processor cores 112 based on execution of the optimistic Gibbs sampler module instructions 116. For example, the optimistic Gibbs sampler module instructions 116, when executed by one of the CPU processor cores, may cause the CPU processor core to select a random variable from a set of random variables. In some cases, the random variable may depend on other random variables of the set of random variables. The instructions may then cause the processor to read values of these other random variables. A sample for the random variable is then generated based on a conditional distribution using the values. A post-sampling conflict for the sample may be detected based on a determination that dependency state data changed after the values were read. Further, based on the post-sampling conflict, the sample may be discarded without committing the sample to an output sample set.

- 5 -

[0016] FIG. 1 also depicts that the processing node 110 may be coupled to a persistent memory 130 (a non-volatile memory, such as flash memory, for example) that may be accessed by the processor core 112 via the memory hub 120. The persistent memory 130 may store persistent copies of the optimistic Gibbs sampler module instructions 116 and the output sample set 118.

[0017] Example operational aspects of a computer device are now discussed in greater detail.

[0018] FIG. 2 is a flowchart illustrating a method 200 for executing a Gibbs sampler using optimistic concurrency control, according to an example. The method 200 may be performed by the modules, components, systems shown in FIG. 1, and, accordingly, is described herein merely by way of reference thereto. For example, in some cases, the method 200 may be performed by an optimistic Gibbs sampler module. An optimistic Gibbs sampler module may be a processor-implemented hardware that utilizes post-sampling conflict detection, such as a processor core 112 executing instructions 116. It will be appreciated that the method 200 may, however, be performed on any suitable hardware.

[0019] The method 200 may begin when, at block 202, an optimistic Gibbs sampler module receives a set of random variables and dependency data specifying dependency relationships between the random variables. By way of example and not limitation, the set of random variables and the dependency data may be a graph $G = \langle V, E \rangle$ that encodes the dependency between random variables. V may be a set of nodes (e.g., $V = \{X_1, X_2, \dots, X_n\}$) in the graph that correspond to the set of random variables. E may be a set of edges. An edge e in E between X_i and X_j may signal that X_i and X_j are dependent. In some cases, a pair of dependent random variables can be connected by an edge or not. In this description, edges are assumed to have no directions and G can contain cycles. These type of graphs are called Markov Random Fields (MRFs). A MRF may encode a joint probability distribution on the random variables in V . Other implementations can be applied to other types of graphs such as Bayesian Networks, Factor Graphs, and the like.

- 6 -

[0020] Each random variable X_i may be associated with a domain D_i which is the set of values X_i can take. Gibbs sampling can be applied to both continuous-valued as well as discrete-valued variables. To simplify the discussion herein, this disclosure may discuss examples in terms of discrete-valued random variables. For example, without loss of generality, this disclosure can assume $D_i = \{0, 1\}$ for all i , in the following discussion. However, other implementations may use continuous-valued random variables.

[0021] At block 204, the optimistic Gibbs sampler module may select a first random variable from the set of random variables. This random variable selected by the optimistic Gibbs sampler module may be the random variable that the optimistic Gibbs sampler module is to generate a sample for during an iteration of the sampling.

[0022] At block 206, based on the dependency data, the optimistic Gibbs sampler module may identify that the selected random variable depends on a second random variable. As described above, when a first random variable depends on a second random variable, the value of the first random variable may depend on the value of the second random variable. In some cases, the optimistic Gibbs sampler module may identify that the selected random variable depends on the other random variables by accessing the graph G to determine which nodes have edges that connect with the node that represents the selected random variable.

[0023] At block 208, the optimistic Gibbs sampler module may generate a sample of the selected random variable according to a conditional distribution. The conditional distribution can be calculated based on values of the random variables on which the selected random variable depends. That is, the optimistic Gibbs sampler module can compute a conditional distribution for the selected random variable using the current values of the neighbors to the selected random variable. In one case, the following equation can be used to calculate the conditional distribution for the selected random variable:

$$p(X_i | X_1 X_2 \dots X_{i-1} X_{i+1} \dots X_n) \propto \prod_{C \in C_{X_i}} \Phi_C$$

- 7 -

[0024] X_i is the selected random variable. $X_1, X_2, \dots, X_{i-1}, X_{i+1}, \dots, X_n$ is the set of values for the neighbors of X_i . The $\propto$ symbol as used herein may signify that the left hand value of the equation is proportional to the right hand side value. C_{X_i} is the set of cliques containing X_i . Φ_c may be a factor potential function of C , which defines how the variables in C are dependent on each other.

[0025] Once a sample is generated for the selected random variable, the optimistic Gibbs sampler module may then begin a verification process. For example, at block 210, after generating the sample, the optimistic Gibbs sampler module may detect a lack of a post-sampling conflict between the sample and the second random variable to which the selected random variable depends. In some cases, a post-sampling conflict may occur when a value of a neighboring random variable changes after the optimistic Gibbs sampler module reads the value for that neighbor but before the optimistic Gibbs sampler module commits the new sample for the selected random variable. In some cases, the optimistic Gibbs sampler module may track changes in dependency state data. Some examples of post-sampling conflict detection are discussed below with reference to FIGS. 3 and 4.

[0026] After detecting the lack of the post-sampling conflict, the optimistic Gibbs sampler module may then, at block 212, commit the generated sample to an output sample set.

[0027] As discussed above with respect to block 210, an optimistic Gibbs sampler module may detect a post-sampling conflict (or the lack thereof) after a sample is generated for the selected random variable. Different techniques for detecting post-sampling conflicts are now discussed in greater detail. For example, one implementation may assign a timestamp to a clique when a sample for a random variable that is a member of the clique is committed. FIG. 3 is a flowchart illustrating a method 300 for using timestamps to detect post-sampling conflicts, according to an example. The method 300 may be performed for an iteration of a sampling loop for the optimistic Gibbs sampler.

- 8 -

[0028] FIG. 3 shows that the method 300 may begin at block 302 when the optimistic Gibbs sampler module calculates a timestamp for the current iteration of the sampling loop. Such a timestamp may be a system clock shared by the different processor cores executing the parallel Gibbs sampler. In other cases, the timestamp may be a global counter accessible to the different processor cores. In yet other cases, the timestamp can be a unique identifier generated at random. With momentary reference to FIG. 2, block 302 may be performed before block 206 is performed. To better differentiate between the timestamp calculated at block 302 with other timestamps used in the method 300, the timestamp calculated at block 302 may be referred to as a “new timestamp.”

[0029] With reference back to FIG. 3, at block 304, the optimistic Gibbs sampler module may obtain, before generating the sample (e.g., block 208 of FIG. 2), timestamps associated with the cliques to which the selected random variable is a member. To better differentiate between the timestamps obtained at block 304 and other timestamps, the foregoing may refer to the timestamps obtained at block 304 as “pre-sampling timestamps.” In some cases, a timestamp may be stored in an entry in a clique timestamp data structure. The “clique timestamp data structure” may be a term that refers to data or logic that maps a given clique with a timestamp. In some cases, the clique timestamp data structure may be a dedicated data structure, such as an array, linked list, search tree, graph, hash table, or the like, that uses the clique (or an identifier associated with the clique) as a key to map to an entry in the clique timestamp data structure that stores the current timestamp for that clique. In other cases, the timestamp for a given clique may be a field of a clique.

[0030] At block 306, the optimistic Gibbs sampler module may obtain, after generating the sample (e.g., block 208 of FIG. 2), timestamps associated with the cliques to which the selected random variable is a member. Similar to block 304, the optimistic Gibbs sampler module may access the clique timestamp data structure for each clique to which the selected random variable is a member. To clarify the

description in the foregoing, the timestamps obtained at block 306 may be referred to as a “post-sampling timestamps.”

[0031] At block 308, the optimistic Gibbs sampler module determines that the pre-sampling timestamps match the post-sampling timestamps. When the pre-sampling timestamps match the post-sampling timestamps, the optimistic Gibbs sampler module may treat this occurrence as a signal that other processor cores have not yet committed a sample that will conflict with the sample currently generated. Accordingly, it is safe for the optimistic Gibbs sampler module to commit the current sample to the output sample set.

[0032] At block 310, the optimistic Gibbs sampler module may then associate the new timestamp with the cliques to which the selected random variable is a member. For example, in some cases, the optimistic Gibbs sampler module may store the new timestamp in entries of the timestamp data structure that map to the cliques.

[0033] The following code sample illustrates pseudocode that illustrates an example implementation of the method 300.

Table 1

Clique Timestamp Approach
Input: Graph $G=\langle V, E \rangle$, set of cliques C_G and their associated factor potentials F, number of iterations T Output: A set of sample tuples $S = \{ \langle t, i, a \rangle \}$ where each sample tuple is a sample generated at iteration t for random variable X_i, whose assignment is a
1. $A \leftarrow$ Initialize A with a random assignment for each variable 2. for all processor cores do in parallel 3. for $t = 1 \dots T$ do 4. get a timestamp τ

- 10 -

```

5.    randomly pick a variable  $X_i$  in  $V$ 
6.     $\Gamma \leftarrow \emptyset$ 
7.    for each clique  $C$  that  $X_i$  is associated with do
8.         $\Gamma \leftarrow \Gamma \cup \{ \langle C, TS(C) \rangle \}$ 
9.    end for
10.   for each neighbor  $X_j$  in  $NE(X_i)$  do
11.       Read( $A_j$ )
12.   end for
13.   compute  $p(X_i | NE(X_i))$ 
14.    $a \leftarrow$  sample according to  $p(X_i | NE(X_i))$ 
15.   // verification phase begins
16.    $\Gamma' \leftarrow \emptyset$ 
17.   for each clique  $C$  that  $X_i$  is associated with do
18.        $\Gamma' \leftarrow \Gamma' \cup \{ \langle C, TS(C) \rangle \}$ 
19.   end for
20.   if  $\Gamma \neq \Gamma'$ 
21.       continue
22.   else
23.       for each clique  $C$   $X_i$  associated with do
24.            $TS(C) \leftarrow \tau$ 
25.       end for
26.        $A_i \leftarrow a$ 
27.        $S \leftarrow S \cup \{ \langle t, i, a \rangle \}$ 
28.   end for
29. end for

```

[0034] In the above pseudocode, A may be a data structure that stores current values for each of the random variables in V . The output sample set A may be initialized such that each of the current values in A are assigned initial values (see

- 11 -

line 1). Lines 3-28 may represent a sampling loop executed independently and in parallel by processor cores of a computer device to generate sample tuples for the random variables V . The sample tuples may be stored in the output sample set S . An optimistic Gibbs sampler module executing on each processing unit may perform the sampling loop T times to generate up to T sample tuples. At line 4, the optimistic Gibbs sampler module may get a new timestamp τ for the current iteration of the sampling loop. At line 5, the optimistic Gibbs sampler module selects a random variable X_i for which a sample is to be generated for iteration t of the sampling loop. At lines 6-9, the optimistic Gibbs sampler module may initialize pre-sampling timestamps Γ for the selected random variable X_i with timestamps presently assigned to each of the cliques C that X_i is a member. At lines 10-14, the optimistic Gibbs sampler module may then generate a Gibbs sample a of the selected random variable. At lines 16-19, the optimistic Gibbs sampler module may initialize a post-sampling timestamps Γ' for the selected random variable X_i with timestamps presently assigned to each of the cliques C that X_i is a member. If, at lines 18-19, the optimistic Gibbs sampler module determines that some of the values in Γ differs from Γ' , the optimistic Gibbs sampler module continues onto the next iteration of the sampling loop without committing the sample a to S . Otherwise, the optimistic Gibbs sampler module can update A with the sampled value a for X_i (line 21) and update S with the sample tuple (line 22).

[0035] FIG. 4 is a flowchart that illustrates a method 400 for detecting post-sampling conflicts in a Gibbs sampler based on comparing neighbor snapshots generated before and after the generation of a sample, according to an example. The method 400 may begin at block 402 when the optimistic Gibbs sampler module uses the dependency data to identify neighbors of the first random variable. In some cases, where the dependency data may be a $N \times N$ matrix DD . N may be the number of random variables. The element $DD_{i,j}$ in DD may have a value 1 if random variable V_i is dependent on random variable V_j ; otherwise, the element $DD_{i,j}$ may have the value 0. According to this implementation, for the selected random variable V_i , the

- 12 -

optimistic Gibbs sampler module reads DD_{ij} (where j is incremented from 1 to N). When the optimistic Gibbs sampler module determines that DD_{ij} has a value of a value of 1, random variable V_j is identified as a neighbor of V_i ; otherwise, V_j is not a neighbor of V_i . It is to be appreciated the dependency data may be implemented using any other suitable structure. For example, it can be appropriate depending on implementation to represent dependencies between the random variables with a collection of tuples T , where each tuple in T represents an edge between two random variables, such as tuple $\langle V_i, V_j \rangle$, which signifies random variable V_i is dependent on V_j . The random variables that are neighbors to the selected random variable V_i may be collectively referred to as $NE(V_i)$.

[0036] At block 404 the optimistic Gibbs sampler module generates a neighbor snapshot. The neighbor snapshot may include sample values of the neighbors of the selected random variable at a time period that precedes the generation of the sample (e.g., before executing block 208 of FIG. 2). To differentiate the neighbor snapshot generated at block 404 from other neighbor snapshots that be my generated by the optimistic Gibbs sampler module, the foregoing may refer to the neighbor snapshot generated at block 404 as a “pre-sampling neighbor snapshot.” To generate the pre-sampling neighbor snapshot, the optimistic Gibbs sampler module may iterate over $NE(V_i)$ and store the current value for each random variable in $NE(V_i)$.

[0037] At block 406 the optimistic Gibbs sampler module generates another neighbor snapshot (i.e., a second neighbor snapshot). This neighbor snapshot may include sample values of the neighbors of the selected random variable at a time period that precedes the generation of the sample (e.g. after block 208 of FIG. 2). To differentiate the neighbor snapshot generated at block 406 from other neighbor snapshots that be my generated by the optimistic Gibbs sampler module, the foregoing may refer to the neighbor snapshot generated at block 406 as a “post-sampling neighbor snapshot.” To generate the post-sampling neighbor snapshot,

- 13 -

the optimistic Gibbs sampler module may iterate over $NE(V_i)$ and store the current value for each random variable in $NE(V_i)$.

[0038] At block 408, the optimistic Gibbs sampler module determines that the pre-sampling neighbor snapshot matches the post-sampling neighbor snapshot. When the pre-sampling neighbor snapshot matches the post-sampling neighbor snapshot, the optimistic Gibbs sampler module may treat this occurrence as a signal that other processor cores have not committed a sample that will conflict with the sample currently generated for the selected random variable. Accordingly, it is safe for the optimistic Gibbs sampler module to commit the current sample for the random variable to the output sample set. To determine whether the two snapshots match each other, the optimistic Gibbs sampler module may compare the value stored for a given random variable in one snapshot with the value stored for that given random variable in the other snapshot. The optimistic Gibbs sampler module may do this for each random variable represented in the snapshots. If each of the values match, then the snapshots may be treated as matching. Otherwise, if one or more values within the snapshots do not match, then the snapshots may be treated as not matching.

[0039] The following code sample illustrates pseudocode that illustrates an example implementation of the method 400.

Table 2

Neighbor Snapshot Approach
Input: Graph $G=\langle V, E \rangle$, set of cliques C_G and their associated factor potentials F, number of iterations T Output: A set of sample tuples $S = \{ \langle t, i, a \rangle \}$, where each sample tuple is a sample generated at iteration t for random variable X_i, whose assignment is a

- 14 -

```

1.  A ← Initialize A with a random assignment for each variable
2.  for all processing cores do in parallel
3.    for t = 1...T do
4.      randomly pick a variable
      Xi in V
5.      α ← ∅
6.      for each neighbor Xj in
      NE(Xi) do
7.        Read(Aj)
8.        α ← α ∪ {<Xj, Aj>}
9.      end for
10.     compute p(Xi | NE(Xi))
11.     a ← sample according to
      p(Xi | NE(Xi))
12.     // verification phase begins
13.     α' ← ∅
14.     for each neighbor Xj in
      NE(Xi) do
15.       Read(Aj)
16.       α' ← α' ∪ {<Xj, Aj>}
17.     end for
18.     if α ≠ α'
19.       continue
20.     else
21.       Ai ← a
22.       S ← S ∪ {<t, i, a>}
23.     end if
24.   end for
25. end for

```

- 15 -



[0040] In the above pseudocode, A may be a data structure that stores current values for each of the random variables in V. A may be initialized such that each of the current values in A is assigned an initial value (see line 1). Lines 3-24 may represent a sampling loop executed by processor cores to generate sample tuples for the random variables V, as may be stored in the output dataset S. The optimistic Gibbs sampler module may the sampling loop T times to generate up to T sample tuples. At line 4, the optimistic Gibbs sampler module selects a random variable X_i for which a sample is to be generated for this iteration. At lines 5-9, the optimistic Gibbs sampler module may initialize a pre-sampling neighbor snapshot α for the selected random variable X_i with values presently stored in A for each neighboring random variable (e.g., X_j). At lines 10-11, the optimistic Gibbs sampler module may then generate a Gibbs sample a of the selected random variable. At lines 14-16, the optimistic Gibbs sampler module may initialize a post-sampling neighbor snapshot α' for the selected random variable X_i with values presently stored (e.g., after generating the Gibbs sample a) in A for each neighboring random variable (e.g., X_j). If, at lines 18-19, the optimistic Gibbs sampler module determines that some of the values in α differs from α' , the optimistic Gibbs sampler module continues, without committing the sample a, onto the next iteration of the sampling loop. Otherwise, the optimistic Gibbs sampler module can update A with the sampled value a for X_i (line 21) and update S with the sample tuple (line 22).

[0041] Some example examples may execute according to a phase that does not apply locks or any other suitable concurrency control before or after generating a sample. These examples can be used to have at least a fraction of the variables get close to their stationary assignments leaving only the problematic vertices (e.g., random variables) to be resolved. Subsequently, these examples then shift to a Gibbs sampling phase that does use concurrency control with the results of the lock-free phase serving as initial values for the Gibbs sampling phase. The overall idea is

to reduce the time taken to “burn-in” the Gibbs sampling. The “burn-in” phase in a conventional Gibbs sampling algorithm refers to the initial set of iterations when the results of the sampling do not truly reflect the underlying distribution. This phase is often discarded when computing statistics used for making inference. By removing locks or other forms of concurrency control, we aim to speed up the “burn-in” period without affecting the overall correctness of the Gibbs sampler. The precise duration of the lock-free phase can be either a user-defined parameter or decided based on the statistics collected from the sampling. If there appears to be a certain degree of convergence in a significant fraction of the variables, then the lock-free phase can be turned off and the phase with concurrency control resumed. For the concurrency control phase, one can use the techniques described above.

Table 3

Lock-Free Phase Approach	
Input: Graph $G=\langle V, E \rangle$, set of cliques C_G and their associated factor potentials F , number of iterations T	
Output: A set of sample tuples $S = \{ \langle t, i, a \rangle \}$, where each sample tuple is a sample generated at iteration t for random variable X_i , whose assignment is a	
1.	$A \leftarrow$ Initialize A with a random assignment for each variable
2.	for all processor cores do in parallel //lock-free phase
3.	for $t=1 \dots T$ do
4.	randomly pick a variable X_i in V
5.	compute $p(X_i \mid NE(X_i))$
6.	$A \leftarrow$ sample according to $p(X_i \mid NE(X_i))$
7.	end for
8.	// Concurrency-control phase
9.	Initialize concurrency control phase with final assignment of lock-free

- 17 -

phase	
10. Proceed with Gibbs sampling with concurrency control	⋮

[0042] In the above pseudocode, A may be a data structure that stores current values for each of the random variables in V. A may be initialized such that each of the current values in A is assigned an initial, possibly random, value (see line 1). Lines 3-7 may represent a sampling loop executed by processor cores of a computer device to generate sample tuples for the random variables V, as may be stored in S. Each optimistic Gibbs sampler module executing on a processing unit may perform the sampling loop T times to generate up to T sample tuples. At line 4, the optimistic Gibbs sampler module selects a random variable X_i for which a sample is to be generated for this iteration. At lines 5-6, the optimistic Gibbs sampler module may then generate a Gibbs sample of the selected random variable. As shown in the above pseudocode, the Gibbs sample is generated and committed without any conflict checks. At line 9, A may again be initialized. However, at line 9, A is initialized with the values generated by the Gibbs samples generated at lines 5 and 6. At line 10, the optimistic Gibbs sampler module may execute a Gibbs sampling that utilizes concurrency controls, such as a lock based approach that acquires locks for neighbor random variables before generating a sample or an optimistic approach, as discussed above.

Claims

What is claimed is:

1. A method comprising:
 - receiving, by a processor, a set of random variables and dependency data specifying dependency relationships between the random variables;
 - selecting, by the processor, a first random variable from the set of random variables;
 - based on the dependency data, identifying, by the processor, that the first random variable depends on a second random variable;
 - generating, by the processor, a first sample of the first random variable according to a conditional distribution, the conditional distribution being calculated based on a value of the second random variable;
 - after generating the first sample, detecting, by the processor, a lack of a post-sampling conflict between the first sample and the second random variable; and
 - committing, by the processor, the first sample to an output sample set.
2. The method of claim 1, wherein the first random variable is a member of a clique, and detecting the lack of the post-sampling conflict comprises:
 - obtaining, before generating the first sample, a first timestamp corresponding to the clique;
 - obtaining, after generating the first sample, a second timestamp corresponding to the clique; and
 - determining that the first timestamp matches the second timestamp.
3. The method of claim 2, further comprising:
 - calculating, prior to generating the first sample, a third timestamp corresponding to the sample; and
 - associate, after determining that the first timestamp matches the second timestamp, the third timestamp with the clique.

- 19 -

4. The method of claim 1, further comprising:
using the dependency data, identifying neighbors of the first random variable;
generating a first neighbor snapshot, the first neighbor snapshot including sample values of the neighbors at a time prior to generating the first sample;
generating a second neighbor snapshot, the second neighbor snapshot including sample values of the neighbors at a time after generating the first; and
determining that the first neighbor snapshot matches the second neighbor snapshot.

5. The method of claim 1, further comprising, before selecting the first random variable:
generating a number of samples for the set of random variables; and
committing, absent conflict checks, the number of samples to the output sample set.

6. A device comprising:
a processor; and
a machine-readable storage device comprising instructions that, when executed, cause the processor to:
select a random variable from a set of random variables;
read first dependency state data of the random variable;
generate a Gibbs sample for the random variable;
read second state data relating to the dependency data of the random variable;
detect a lack of a post-sampling conflict for the Gibbs sample based on a comparison of the first state and second state; and
commit the Gibbs sample to an output sample set.

7. The device of claim 6, further comprising another processor, wherein

- 20 -

the instructions cause, in parallel with the processor, the another processor to:

- select another random variable from the set of random variables;
- read third dependency state data of the another random variable;
- generate another Gibbs sample for the another random variable;
- read fourth state data relating to the dependency data of the another random variable;
- detect a post-sampling conflict for the another Gibbs sample based on a comparison of the third state and fourth state; and
- discard the another Gibbs sample without committing the another Gibbs sample to the output sample set.

8. The device of claim 6, wherein the first dependency state data is a first neighbor snapshot of values of neighbors of the random variable, and the second dependency state data is a second neighbor snapshot of values of the neighbors of the random variable.

9. The device of claim 6, wherein the first dependency state data is a first timestamp for a clique that the random variable is a member, and the second dependency state data is a second timestamp for the clique.

10. The device of claim 6, wherein the instructions, when executed, further cause the processor to:

- generate a number of samples for the set of random variables; and
- commit, absent conflict checks, the number of samples to the output sample set.

11. A machine-readable storage device comprising instructions that, when

- 21 -

executed, cause a processor to:

- select a random variable from a set of random variables, the random variable depends on other random variables in the set of random variables;

- read values of the other random variables;

- generate a sample for the random variable based on a conditional distribution using the values;

- detect a post-sampling conflict for the sample based on a determination that dependency state data changed after the values were read; and

- based on the post-sampling conflict, discard the sample without committing the sample to an output sample set.

12. The machine-readable storage device of claim 11, wherein the dependency state data includes: a first timestamp for a clique that the random variable is a member and a second timestamp for the clique, and the instructions, when executed, cause the processor to:

- compare the first timestamp with the second timestamp.

13. The machine-readable storage device of claim 11, wherein the dependency state data includes is a first neighbor snapshot of values of neighbors of the random variable and a second neighbor snapshot of the values of the neighbors of the random variable.

14. The machine-readable storage device of claim 13, wherein the instructions, when executed, cause the processor to:

- calculate the first neighbor snapshot of values of neighbors before the sample is generated; and

- calculate the second neighbor snapshot of the values of the neighbors after the sample is generated.

- 22 -

15. The machine-readable storage device of claim 11, wherein the instructions, when executed, cause the processor to:
generate a number of samples for the set of random variables; and
commit, absent conflict checks, the number of samples to the output sample set.

1/4

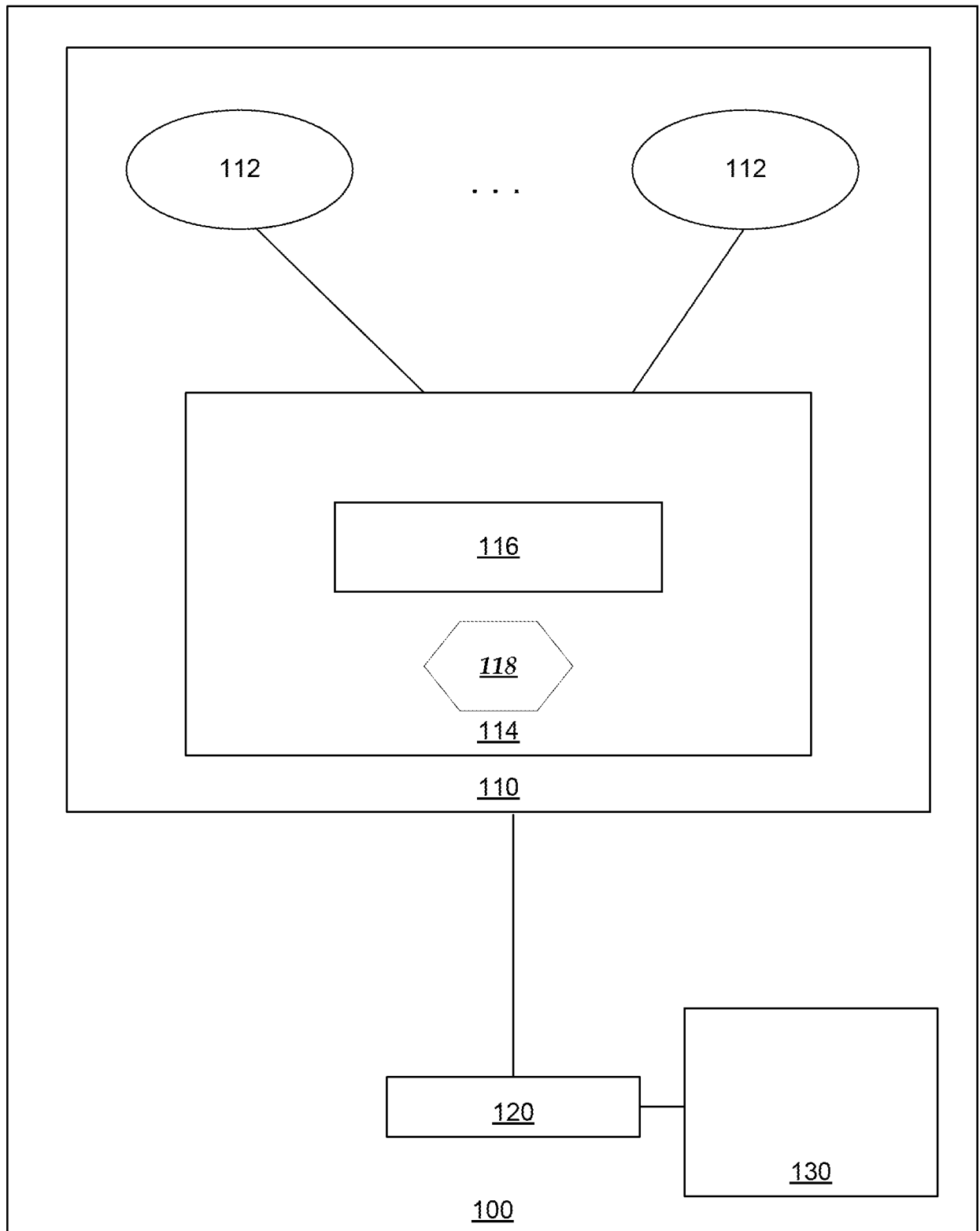


FIG. 1

2/4

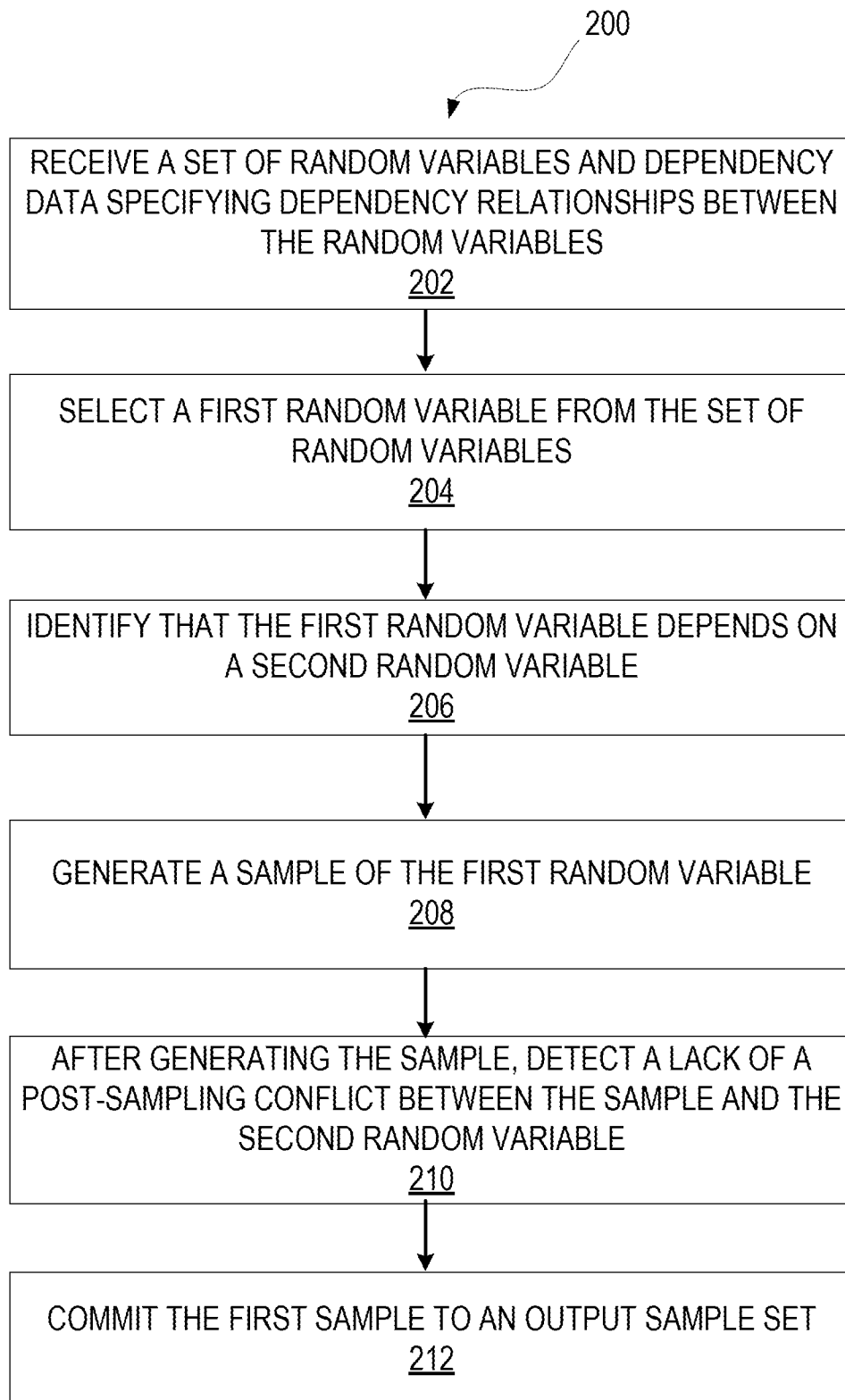
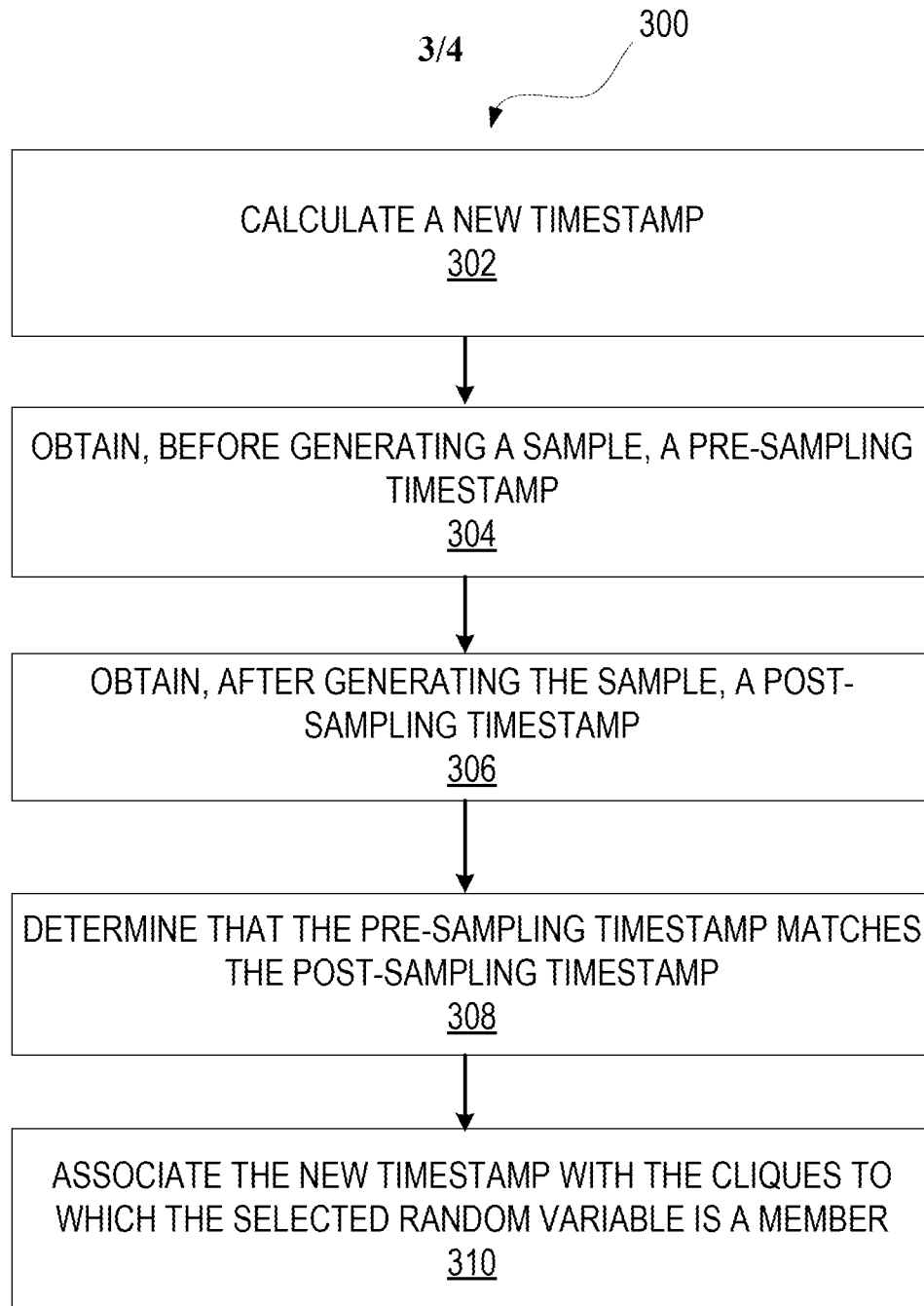


FIG. 2

*FIG. 3*

4/4

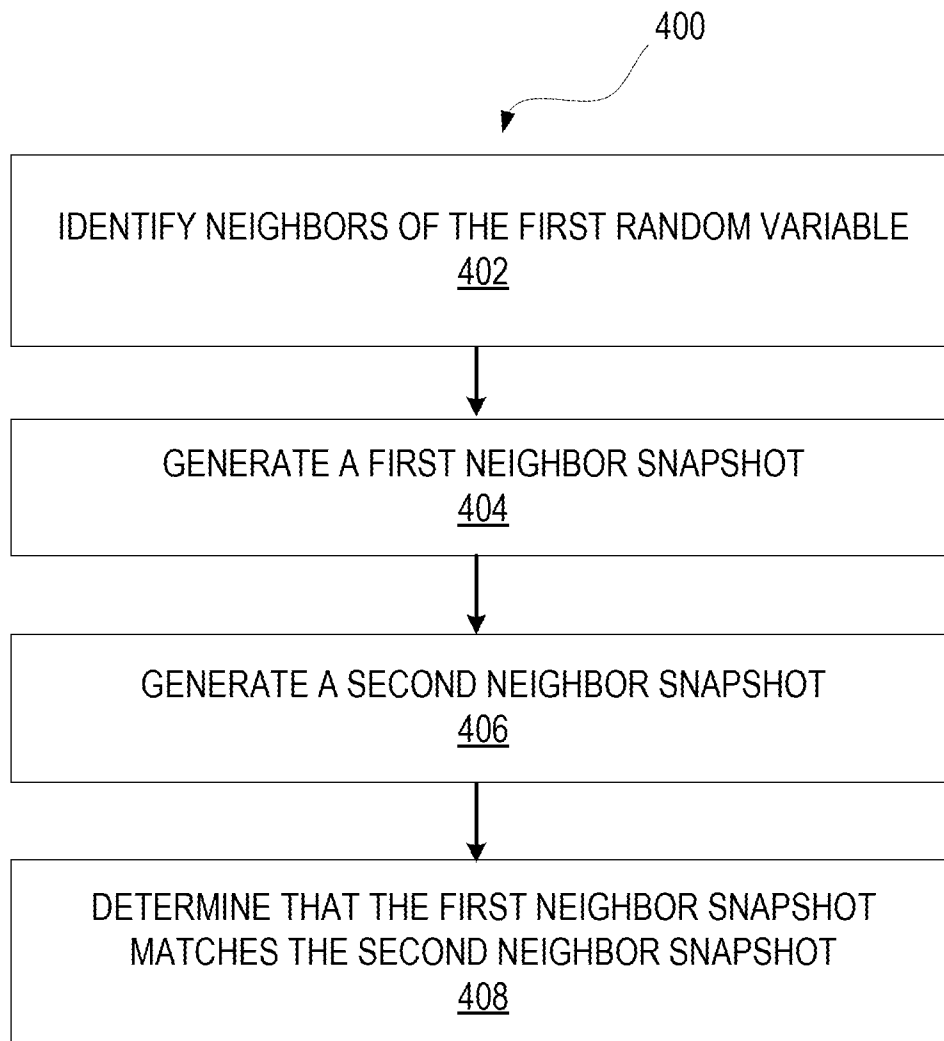


FIG. 4

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/016695**A. CLASSIFICATION OF SUBJECT MATTER****G06F 17/30(2006.01)i**

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 17/30; G06F 17/10; G06F 19/00; G06F 17/50; G06N 99/00; G06N 5/04; G06F 11/36; G06F 11/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS(KIPO internal) & Keywords: random variable, dependency, sample, distribution, conflict, commit, and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2011-0113288 A1 (UMASUTHAN RAMAKRISHNAN et al.) 12 May 2011 See paragraphs [0005]–[0011]; claims 4–5; and figures 1–2.	1–15
A	US 2009-0228238 A1 (VIKASH KUMAR MANSINGHKA et al.) 10 September 2009 See paragraphs [0014]–[0018] and [0190]–[0195]; and figure 1.	1–15
A	US 2014-0122393 A1 (KENJI AOKI) 01 May 2014 See paragraphs [0010]–[0017] and figures 1–2.	1–15
A	WO 2012-055045 A2 (SOLIDO DESIGN AUTOMATION INC.) 03 May 2012 See paragraphs [0017]–[0023] and figure 2.	1–15
A	US 2008-0235172 A1 (MICHAEL TODD ROSENSTEIN et al.) 25 September 2008 See paragraphs [0005]–[0010] and figure 1.	1–15



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

07 September 2015 (07.09.2015)

Date of mailing of the international search report

08 September 2015 (08.09.2015)

Name and mailing address of the ISA/KR

International Application Division
Korean Intellectual Property Office
189 Cheongsu-ro, Seo-gu, Daejeon Metropolitan City, 35208,
Republic of Korea

Facsimile No. +82-42-472-7140

Authorized officer

NHO, Ji Myong

Telephone No. +82-42-481-8528



INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2015/016695

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2011-0113288 A1	12/05/2011	US 8161329 B2	17/04/2012
US 2009-0228238 A1	10/09/2009	CN 102016616 A	13/04/2011
		EP 2260421 A2	15/12/2010
		US 8352384 B2	08/01/2013
		WO 2009-111559 A2	11/09/2009
		WO 2009-111559 A3	11/11/2010
US 2014-0122393 A1	01/05/2014	WO 2012-176863 A1	27/12/2012
WO 2012-055045 A2	03/05/2012	DE 112011103563 T5	22/08/2013
		DE 112011103563 T8	21/11/2013
		JP 2013-542526 A	21/11/2013
		KR 10-2014-0024252 A	28/02/2014
		TW 201237647 A	16/09/2012
		US 2013-0226544 A1	29/08/2013
		WO 2012-055045 A3	21/06/2012
US 2008-0235172 A1	25/09/2008	US 8200600 B2	12/06/2012