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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 April 2017 (20.04.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/065795 A1

(51) International Patent Classification:
G06F 17/00 (2006.01)

(21) International Application Number:
PCT/US2015/055883

(22) International Filing Date:
16 October 2015 (16.10.2015)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP** [US/US]; 11445 Compaq Center Drive W., Houston, Texas 77070 (US).

(72) Inventors: **KAFAI, Mehran**; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US). **DIMITRIADOU, Kyriaki**; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US). **MITCHELL, April Slayden**; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US).

(74) Agents: **DAS, Manav** et al.; Hewlett Packard Enterprise, Intellectual Property Adminsitration, 3404 E. Harmony Road, Mail Stop 79, Fort Collins, Colorado 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, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

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

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

Published:

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

(54) Title: INCREMENTAL UPDATE OF A NEIGHBOR GRAPH VIA AN ORTHOGONAL TRANSFORM BASED INDEXING

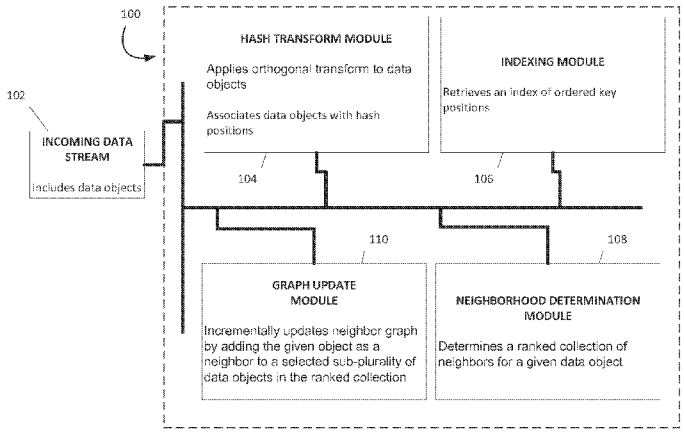


Fig. 1

(57) Abstract: Incremental update of a neighbor graph via an orthogonal transform based indexing is disclosed. One example is a system including a hash transform module to apply an orthogonal transform to a data object in a data stream, and to associate the data object with a collection of ordered hash positions. An indexing module retrieves an index of ordered key positions, where each key position is indicative of data objects in the data stream that have a hash position at the key position. A neighbor determination module determines a ranked collection of neighbors for the data object in a neighbor graph, where the ranking is based on the index. A graph update module incrementally updates the neighbor graph by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection.

WO 2017/065795 A1

INCREMENTAL UPDATE OF A NEIGHBOR GRAPH VIA AN ORTHOGONAL TRANSFORM BASED INDEXING

Background

[0001] Big data is data that generally exceeds the processing capacity of conventional systems. Methods for analyzing big data are of great interest in many applications, including similarity search, collaborative filtering, data compression, retrieval, clustering, and classification. Graphs may be used to represent data. For example, a nearest neighbor graph may connect each data object to data objects that are nearest (or most similar) to the data object.

Brief Description of the Drawings

[0002] Figure 1 is a functional block diagram illustrating one example of a system for incremental update of a neighbor graph via an orthogonal transform based indexing.

[0003] Figure 2 is a block diagram illustrating one example of a computer readable medium for incremental update of a neighbor graph via an orthogonal transform based indexing.

[0004] Figure 3 is a flow diagram illustrating one example of a method for incremental update of a neighbor graph via an orthogonal transform based indexing.

[0005] Figure 4 is a flow diagram illustrating one example of a method for providing an incremental update of a neighbor graph via an orthogonal transform based indexing.

[0006] Figure 5 is a flow diagram illustrating one example of a method for maintaining a database of ranked collections for an incremental update of a neighbor graph via an orthogonal transform based indexing.

[0007] Figure 6 is a flow diagram illustrating one example of a method an index for an incremental update of a neighbor graph via an orthogonal transform based indexing.

Detailed Description

[0008] Big data may be defined as data that exceeds the processing capacity of conventional systems. Methods for analyzing such data continue to grow in prominence. Graph techniques are often applied to represent and efficiently analyze big data. One such graph representation is a *k*-nearest neighbor ("kNN") graph. A kNN graph comprises of nodes that are data objects, and edges that connect a data object to *k* proximal data objects, where proximity between data objects may be based on an appropriate similarity.

[0009] A kNN graph may help represent and analyze large volumes of data by revealing nearest matches to any data object. kNN graphs are extensively used in areas such as machine learning, data mining, computer vision and image processing. Applications that utilize kNN graphs include but are not limited to similarity search, collaborative filtering, data compression, retrieval, clustering, and classification. For example, large Enterprise data warehousing applications may use collaborative filtering to sort and/or search content relevant to a particular topic thread. As another example, an online retailer may use a similarity search to make product recommendations based on a particular customer's recent purchases. Also, for example, a security firm may use nearest neighbor matches to identify known persons from live video feeds.

[0010] Generally, the challenge of these applications is to maintain performance efficiency as data grows. In some examples, the data may be large volumes of rapidly streaming data, adding to the challenge of maintaining both speed and efficiency. For kNN graphs, costs associated with graph construction, for example in terms of time and resources, are considerably high for large data sets. Additionally, as a kNN graph is continually updated based on incoming data, such costs may increase considerably.

[0011] Nearest neighbor graph constructions generally use various methods such as recursive Lanczos bisection, Morton ordering, or local search. Generally, such approaches construct the nearest neighbor graph by comparing data objects directly. Approximate algorithms such as locality sensitive hashing may reduce the comparison space, but data objects in the smaller comparison space are compared directly to each other. This is generally not effective for high velocity, high volume

data streams. Some hash-based techniques may be used to update the kNN graph, but these hashing techniques often display anomalies due to a distance metric distortion.

[0012] Accordingly, there is a need for a kNN graph construction and updating process that may compute and update nearest neighbors in an efficient manner, especially for high velocity, high volume data streams. As described herein, an orthogonal transform based indexing system may be used to efficiently compute the nearest neighbors of a data object in a kNN graph, and then use a local search to update the data objects in the kNN graph that are likely to change after a new data object is inserted into the kNN graph. Such a system is well-adapted for streaming data scenarios and scales well with high-dimensional data as well.

[0013] As described in various examples herein, incremental update of a neighbor graph via an orthogonal transform based indexing is disclosed. One example is a system including a hash transform module to apply an orthogonal transform to a data object in a data stream, and to associate the data object with a collection of ordered hash positions. An indexing module retrieves an index of ordered key positions, where each key position is indicative of data objects in the data stream that have a hash position at the key position. A neighbor determination module determines a ranked collection of neighbors for the data object in a neighbor graph, where the ranking is based on the index. A graph update module incrementally updates the neighbor graph by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, where the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection. Such an incremental construction and update of the kNN graph of the data stream may be performed without directly computing a similarity/distance between pairs of data objects.

[0014] In the following detailed description, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration specific examples in which the disclosure may be practiced. It is to be understood that other examples may be utilized, and structural or logical changes may be made without departing from the scope of the present disclosure. The following detailed description, therefore, is not to be taken in a limiting sense, and the scope of the

present disclosure is defined by the appended claims. It is to be understood that features of the various examples described herein may be combined, in part or whole, with each other, unless specifically noted otherwise.

[0015] Figure 1 is a functional block diagram illustrating one example of a system 100 for incremental update of a neighbor graph via an orthogonal transform based indexing. System 100 includes a hash transform module 104, an indexing module 106, a neighborhood determination module 108, and a graph update module 110. The hash transform module 104 may apply, via a processor, an orthogonal transform to a data object in an incoming data stream 102, and may associate the data object with a subset of U ordered hash positions.

[0016] A flow rate f for the incoming data stream 102, say S, may be measured as data objects received per second. In other words, every second, f number of data objects arrive via incoming data stream S. Accordingly, hash transforms may generally be associated at a similar rate. As described herein, hash transform module 104 may take the data object (e.g. a numerical N-dimensional vector) in the incoming data stream 102, and may associate the data object with a subset of U ordered hash positions. In some examples, the subset may have a fixed size H. A hash position generally refers to a hash symbol located at an integer from the set $\{1, 2, \ldots, U\}$, where U is generally much larger than N, and N is much larger than H. Hashes may be generally used to provide a representation of the data object. Such a hash representation may be utilized for efficient comparisons between pairs of data objects. Generally, U may be a power of 2. For example, one instance may be $H = 100$, $N = 6000$, and $U = 2^{18}$. The H hashes may be based on largest H coefficients provided by the orthogonal transform. Accordingly, an N-dimensional object may be projected onto a lower H-dimensional object.

[0017] In some examples, the incoming data stream 102 may include a plurality of vectors with numerical, real-valued components. System 100 may be provided with values for H and U. The integer H may be experimentally determined based on the type and number of data objects in the incoming data stream 102. Generally, U is a very large integer relative to H. In one example, U is a power of 2. The elements of system 100 may be implemented, for example, in software.

[0018] In some examples, the orthogonal transform may be a Walsh-Hadamard transform ("WHT"). A WHT is an orthogonal, non-sinusoidal transform that takes a signal as input and outputs a set of basis functions. The output functions are known as Walsh functions. A Walsh function takes two values: +1 and -1. Generally, performing an orthogonal transform on the incoming data stream 102 provides a set of coefficients associated with the incoming data stream 102. For example, after application of the Walsh-Hadamard transform to the data object, the hash transform module 104 may provide a collection of coefficients $c_1, c_2, \ldots, c_k$. Alternative forms of orthogonal transforms may be utilized as well. For example, a cosine transform may be utilized.

[0019] In some examples, the *H* largest coefficients, $c_{n_1}, c_{n_2}, \ldots, c_{n_H}$ may be selected as the subset of hashes associated with the data object, say A. Table 1 illustrates an example association of data objects A, B, and C, with sets of hashes:

Data Term	Set of Hashes
A	{1, 5, 9, 13, 16}
B	{2, 3, 4, 7, 13}
C	{1, 5, 7, 8, 11}

Table 1

[0020] System 100 includes the indexing module 106 to retrieve, via the processor, an index of *U* ordered key positions, where each key position is indicative of received data objects of the incoming data stream 102 that have a hash position at the key position. Table 2 illustrates an example index with $U = 2^4$ key positions {1, 2, …, 16} with data objects A, B, and C, based on sets of *H* = 5 hashes in Table 1:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A				A				A				A			A
	B	B	B			B						B			
C				C		C	C			C					

Table 2

[0021] As illustrated, ordered key positions 1 and 5 are indicative of received data objects A and C since these data objects have hashes at the hash positions 1 and 5, as illustrated in Table 1. Likewise, ordered key position 13 is indicative of

received data objects A and B since these data objects have hashes at the hash position 13, as illustrated in Table 1.

[0022] In some examples, the indexing module 106 may update the index by inserting, via the processor, a unique identifier associated with the data object into each key position of the index that corresponds to a hash position for the data object. For example, data objects B and C may have been received by system 100 prior to receiving the data object A. Accordingly, the indexing module 106 may have updated the index by inserting unique identifiers associated with the data object A. For example, the data object A may have hashes at hash positions 1, 5, 9, 13, and 16, as illustrated in Table 1. Accordingly, the indexing module 106 may update the index by inserting unique identifiers associated with the data object A at key positions 1, 5, 9, 13, and 16, as illustrated in Table 2.

[0023] A similarity measure between two data objects may be determined based on a number of common hashes. This provides an approximate measure of nearest neighbors. An approximate nearest neighbor ("ANN") search may then be performed. For example, a received data object may be mapped to a set of hashes, and this set of hashes may be utilized to search for neighbors in the incoming data stream 102 based on the similarity measure. The similarity measure may be based on a number of overlaps between respective sets of hashes, and is indicative of proximity of the pair of data objects. Table 3 illustrates an example determination of similarity measures for pairs formed from the data objects A, B, and C:

Data Object Pair: (X,Y)	Similarity Measure: S(X,Y)
(A,B)	S(A,B) = 1
(A,C)	S(A,C) = 2
(B,C)	S(B,C) = 1

Table 3

[0024] As illustrated in Table 2, the data objects A and B have key position 13 in common in their respective sets of hashes. Accordingly, the similarity measure for the pair (A,B), denoted as S(A,B) may be determined to be 1. Also, for example, as illustrated in Table 2, the data objects A and C have key positions 1 and 5 in

common in their respective sets of hashes. Accordingly, the similarity measure for the pair (A,C), denoted as S(A,C) may be determined to be 2. As another example, as illustrated in Table 2, the data objects B and C have key position 7 in common in their respective sets of hashes. Accordingly, the similarity measure for the pair (B,C), denoted as S(B,C) may be determined to be 1.

[0025] System 100 includes the neighborhood determination module 108 to determine, via the processor, a ranked collection of neighbors for the data object in a neighbor graph, where the ranking is based on the index. A neighbor graph, such as a kNN graph for stream S, is a time-varying directed graph where each vertex (representing an object) is connected to similar vertices. In some examples, each vertex may be connected to its top-k most similar vertices via a directed edge. For streaming data scenarios with a high flow rate, it may be desirable to construct an approximate kNN graph and update it after every data object arrival/insertion. For a flow rate of f, this may generally involve f updates per second. As described herein, in some examples, an approximate nearest neighbor list of k nearest neighbors for the data object may be maintained. Such a list may be incrementally updated with the arrival of each new data object in the incoming data stream 102.

[0026] Generally, when a new data object arrives (say n-th data object), a naïve approach may be to update all previous (n-1) lists for the (n-1) data objects received prior to the n-th data object. This may lead to $n\frac{(n-1)}{2}$ nearest neighbor list updates or $\vartheta(n^2)$ complexity for updating the kNN graph. This estimate does not take into account the time taken to retrieve the nearest neighbors.

[0027] In some examples, the indexing module 106 may maintain a database of ranked collections associated with data objects in the incoming data stream.102. In some examples, the ranked collection of neighbors for the data object may comprise top k nearest neighbors (e.g., based on the similarity measure) determined based on a count of overlapping hash positions in the updated index. In some examples, the ranked collection of neighbors for the data object may comprise a predetermined number of neighbors. In some examples, the ranked collection of neighbors for the data object may comprise neighbors that are approximately close to the data object. For example, neighbors may be determined to be ones that have similarity measures in the range 3 to 5. An

example illustration of the ranked collection is described herein with respect to a next data object.

[0028] In some examples, the hash transform module 104 may apply a hash transform to the next data object, where the next data object is received after the data object, and the neighbor determination module 108 may determine a new ranked collection of neighbors for the next data object based on the updated index. For example, the next data object Q may be received after the data object A, which, in turn, may be received after the received data objects B and C. The hash transform module 104 may associate the next data object Q with a subset of U ordered hash positions, say {1, 5, 7, 9, 13}. Table 4 illustrates an example updated index with $U = 2^4$ key positions {1, 2, …, 16} with data objects A, B, C, and Q:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A				A				A				A			A
	B	B	B			B						B			
C				C		C	C			C					
Q				Q		Q		Q				Q			

Table 4

[0029] Table 5 illustrates an example list of terms associated with the next data object Q illustrated in Table 4, and the corresponding similarity measures.

1	5	6	7	9	10	13	Similarity Measures
A	A			A		A	S(Q,A) = 4
			B			B	S(Q,B) = 2
C	C		C				S(Q,C) = 3

Table 5

[0030] As illustrated, the set of hashes associated with the query term Q may be compared with the indexed data objects illustrated in Table 2. Based on such a comparison, index position 1 is indicative of a hash position associated with Q and index position 1 is also indicative of hash positions associated with data objects A and C. Also, for example, index position 5 is indicative of a hash position associated with Q, and index position 5 is also indicative of hash positions associated with data objects A and C. As another example, index position 13 is indicative of a hash position associated with Q, and index position 13 is also indicative of hash positions associated with data objects A and B. A frequency of

occurrence for A may be determined to be 4. This is also the similarity measure for the pair Q and A. A frequency of occurrence for B may be determined to be 2. This is also the similarity measure for the pair Q and B. A frequency of occurrence for C may be determined to be 3. This is also the similarity measure for the pair Q and C.

[0031] As described herein, the neighbor determination module 108 may determine a new ranked collection of neighbors for the next data object based on the updated index (e.g., as illustrated in Table 4). For example, based on a similarity measure of the next data object with each data object, the neighbor determination module 108 may determine a new ranked collection of neighbors for the next data object. Based on the example illustrated in Table 5, the new ranked collection of neighbors for the next data object Q may be determined to be A, C, and B.

[0032] System 100 includes the graph update module 110 to incrementally update the neighbor graph via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, where the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection. The term "incremental update", as used herein, generally refers to an iterative update of the neighbor graph as data objects arrive. As described herein, such an incremental update may be based on a local determination of neighborhood relations. For example, although an incoming data object may be a nearest neighbor for a large number of previously received data objects, an appropriate sub-collection of these received data objects may be selected for an incremental update, the neighbor graph may be updated to reflect a change in neighborhoods for the selected data objects.

[0033] In some examples, the neighborhood comparisons may be based on comparisons between the ranked collection of neighbors for the data object and additional ranked collections of neighbors for data objects in the ranked collection. For example, when a data object x is received by system 100, the neighbor determination module 108 may determine a ranked collection $B(x)$ associated with the data object x. In some examples, a collection of top-*m* most similar data objects (e.g., highest similarity measures with respect to the data object x) may be selected from ranked collection $B(x)$, say $x_1, x_2, \ldots, x_m$. In some examples, each of these

data objects may be associated with respective additional ranked collections, $B(x_1), B(x_2), \ldots, B(x_m)$.

[0034] In some examples, the additional ranked collection $B(x_1)$ corresponding to the data object x_1 may comprise neighbor $x_{11}, x_{12}, \ldots, x_{1k}$, where x_{11} is closest to x_1, and x_{1k} is the k-th furthest away from x_1. A notion of distance-based terms "close" and "further", as used herein, may be defined quantitatively with respect to, for example, the similarity measures determined herein. For example, two data objects that have a larger count of overlapping hashes (e.g., a larger similarity measure) may be determined to be closer than two data objects that have a smaller count of overlapping hashes (e.g., a smaller similarity measure). For example, with reference to Table 3, the data objects A and C are closer (with similarity measure 2) than the data objects A and B (with similarity measure 1).

[0035] Accordingly, the graph update module 110 may compare the distance from x_1 to the data object in $B(x_1)$ that is furthest away from x_1, for example, x_{1k}, with the distance from x_1 to x. If the distance from x_1 to x is less than the distance from x_1 to x_{1k}, then the graph update module 110 may incrementally update the neighbor graph to include x in the ranked collection $B(x_1)$ corresponding to the data object x_1, and add x as a neighbor for x_1. In some examples, the graph update module 110 may perform such a comparison for each data object in $B(x_1)$. Accordingly, an updated ranked collection may be determined for x_1.

[0036] In some examples, the graph update module 110 may repeat, concurrently, and/or iteratively, the process described with reference to $B(x_1)$ for each of the top-*m* data objects $x_1, x_2, \ldots, x_m$ in the ranked collection $B(x)$ associated with the data object x. The number of neighbor comparisons may be further reduced by using a search algorithm (e.g., a binary search) and/or an efficient data structure to store the neighbor graph, the index, and/or the ranked collections for each data object.

[0037] A critical parameter for the process, as described herein, is the parameter *m*, corresponding to the top-*m* data objects $x_1, x_2, \ldots, x_m$ in the ranked collection $B(x)$. In some examples, a value for the parameter *m* may be determined by experimenting with an accuracy of incremental update of neighbor graphs determined based on different values of the parameter *m*.

[0038] As described herein, the incremental update process does not compute distances and/or similarity measures between every possible pairs of potential and/or actual neighbors. In terms of computational complexity based on a number of operations performed, the complexity of this incremental update process may be approximately determined as $\vartheta(mn)$, as compared to a complexity of $\vartheta(n^2)$ to update the entire kNN graph. Generally, a cost of a single operation may be less expensive than existing approaches that directly compare all or a subset of the data points. The process described herein does not depend on a dimensionality of x, thereby enhancing run-time efficiencies.

[0039] In some examples, the graph update module 110 may provide, via the processor to a computing device, an incrementally updated computer generated visual rendition of the neighbor graph for the incoming data stream 102. For example, as each data object is received by system 100 and processed as described herein, the graph update module 110 may retrieve the updated ranked collections, and display the data objects in the neighbor graph by adding nodes for the data objects, and by adding vertices between data objects to indicate updated neighbor relations between data objects. For example, when the data object x is added to the ranked collection $B(x_1)$ for x_1 via removal of a previous neighbor x_{1k}, the graph update module 110 may generate an incrementally updated visual rendition of the neighbor graph by adding a node representing x to the neighbor graph, and by adding an edge between x and x_1, indicative of an addition of x as a neighbor for x_1. Also, for example, the graph update module 110 may generate an incrementally updated visual rendition of the neighbor graph by removing an edge between x_{1k} and x_1, indicative of a deletion of x_{1k} as a neighbor for x_1.

[0040] The components of system 100 may be computing resources, each including a suitable combination of a physical computing device, a virtual computing device, a network, software, a cloud infrastructure, a hybrid cloud infrastructure that may include a first cloud infrastructure and a second cloud infrastructure that is different from the first cloud infrastructure, and so forth. The components of system 100 may be a combination of hardware and programming for performing a designated visualization function. In some instances, each component may include a processor and a memory, while programming code is

stored on that memory and executable by a processor to perform a designated function.

[0041] For example, the hash transform module 104 is a combination of hardware and programming to apply the orthogonal transform to the data objects in the incoming data stream 102 via the processor. Also, for example, the hash transform module 104 may include software programming to generate the hash positions for the data objects, and may be communicatively linked to a hardware storage server, and/or a cloud server to store the hash positions for the data objects.

[0042] Likewise, the indexing module 106 is a combination of hardware and programming to retrieve, via the processor, the index of U ordered key positions. For example, the indexing module 106 may include software programming to automatically update the index by inserting, via the processor, unique identifiers associated with each data object into each key positions of the index that correspond to hash positions for the data object. Also, for example, the indexing module 106 may include hardware to physically store, for example, the updated index. Also, for example, the indexing module 106 may include a combination of hardware and software programming to dynamically interact with the other components of system 100.

[0043] In like manner, the neighborhood determination module 108 is a combination of hardware and programming. For example, the neighborhood determination module 108 may include software programming instructions to determine the ranked collections of neighbors for the data objects. Also, for example, the neighborhood determination module 108 may include hardware to physically store, for example, the ranked collections. The neighborhood determination module 108 may include a combination of hardware and software programming to dynamically interact with the other components of system 100.

[0044] Also, for example, the graph update module 110 is a combination of hardware and programming. For example, the graph update module 110 may include software programming instructions to update the neighbor graph, and may include hardware to physically store, for example, the updated neighbor graph. As another example, the graph update module 110 may include software programming instructions to generate an incrementally updated computer

generated visual rendition of the neighbor graph, and may include hardware to physically display the visual rendition of the neighbor graph. The graph update module 110 may include a combination of hardware and software programming to dynamically interact with the other components of system 100.

[0045] Generally, as described herein, the components of system 100 may include software programming and physical networks to be communicatively linked to other components of system 100. In some instances, the components of system 100 may include a processor and a memory, while programming code is stored and on that memory and executable by a processor to perform designated functions.

[0046] A computing device, as used herein, may be, for example, a web-based server, a local area network server, a cloud-based server, a notebook computer, a desktop computer, an all-in-one system, a tablet computing device, a mobile phone, an electronic book reader, or any other electronic device suitable for provisioning a computing resource to perform a unified visualization interface. The computing device may include a processor and a computer-readable storage medium.

[0047] Figure 2 is a block diagram illustrating one example of a computer readable medium for incremental update of a neighbor graph via an orthogonal transform based indexing. Processing system 200 includes a processor 202, a computer readable medium 208, input devices 204, and output devices 206. Processor 202, computer readable medium 208, input devices 204, and output devices 206 are coupled to each other through a communication link (e.g., a bus).

[0048] Processor 202 executes instructions included in the computer readable medium 208. Computer readable medium 208 includes orthogonal transform application instructions 210 to apply, via a processor, an orthogonal transform to a data object in an incoming data stream 102, and associating the data object with a subset of U ordered hash positions.

[0049] Computer readable medium 208 includes index retrieval instructions 212 to retrieve, via the processor, an index of U ordered key positions, wherein each key position is indicative of received data objects of the incoming data stream 102 that have a hash position at the key position.

[0050] Computer readable medium 208 includes index update instructions 214 to update the index by inserting, via the processor, a unique identifier associated with the data object into each key position of the index that corresponds to a hash position for the data object.

[0051] Computer readable medium 208 includes ranked collection determination instructions 216 to determine, via the processor, a ranked collection of neighbors for the data object in a neighbor graph, wherein the ranking is based on the updated index.

[0052] Computer readable medium 208 includes graph update instructions 218 to incrementally update the neighbor graph via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, wherein the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection.

[0053] Computer readable medium 208 includes graph rendition instructions 220 to provide, via the processor to a computing device, an incrementally updated computer generated visual rendition of the neighbor graph for the incoming data stream 102.

[0054] In some examples, computer readable medium 208 includes instructions to maintain an incrementally updated database of ranked collections of neighbors for data objects in the incoming data stream 102.

[0055] Input devices 204 include a keyboard, mouse, data ports, and/or other suitable devices for inputting information into processing system 200. In some examples, input devices 204, such as a computing device, are used to receive the incoming data stream 102. Output devices 206 include a monitor, speakers, data ports, and/or other suitable devices for outputting information from processing system 200. In some examples, output devices 206 are used to provide the incrementally updated computer generated visual rendition of the neighbor graph for the incoming data stream 102.

[0056] As used herein, a "computer readable medium" may be any electronic, magnetic, optical, or other physical storage apparatus to contain or store information such as executable instructions, data, and the like. For example, any computer readable storage medium described herein may be any of Random

Access Memory (RAM), volatile memory, non-volatile memory, flash memory, a storage drive (e.g., a hard drive), a solid state drive, and the like, or a combination thereof. For example, the computer readable medium 208 can include one of or multiple different forms of memory including semiconductor memory devices such as dynamic or static random access memories (DRAMs or SRAMs), erasable and programmable read-only memories (EPROMs), electrically erasable and programmable read-only memories (EEPROMs) and flash memories; magnetic disks such as fixed, floppy and removable disks; other magnetic media including tape; optical media such as compact disks (CDs) or digital video disks (DVDs); or other types of storage devices.

[0057] As described herein, various components of the processing system 200 are identified and refer to a combination of hardware and programming configured to perform a designated visualization function. As illustrated in Figure 2, the programming may be processor executable instructions stored on tangible computer readable medium 208, and the hardware may include processor 202 for executing those instructions. Thus, computer readable medium 208 may store program instructions that, when executed by processor 202, implement the various components of the processing system 200.

[0058] Such computer readable storage medium or media is (are) considered to be part of an article (or article of manufacture). An article or article of manufacture can refer to any manufactured single component or multiple components. The storage medium or media can be located either in the machine running the machine-readable instructions, or located at a remote site from which machine-readable instructions can be downloaded over a network for execution.

[0059] Computer readable medium 208 may be any of a number of memory components capable of storing instructions that can be executed by Processor 202. Computer readable medium 208 may be non-transitory in the sense that it does not encompass a transitory signal but instead is made up of one or more memory components configured to store the relevant instructions. Computer readable medium 208 may be implemented in a single device or distributed across devices. Likewise, processor 202 represents any number of processors capable of executing instructions stored by computer readable medium 208. Processor 202

may be integrated in a single device or distributed across devices. Further, computer readable medium 208 may be fully or partially integrated in the same device as processor 202 (as illustrated), or it may be separate but accessible to that device and processor 202. In some examples, computer readable medium 208 may be a machine-readable storage medium.

[0060] Figure 3 is a flow diagram illustrating one example of a method for incremental update of a neighbor graph via an orthogonal transform based indexing. In some examples, such an example method may be implemented by a system such as, for example, system 100 of Figure 1.

[0061] At 300, an incoming data stream 102 comprising data objects is received via a processor.

[0062] At 302, an orthogonal transform is applied to a data object in the incoming data stream 102.

[0063] At 304, the data object is associated with a subset of U ordered hash positions.

[0064] At 306, an index of U ordered key positions is retrieved via the processor, where each key position is indicative of received data objects of the incoming data stream 102 that have a hash position at the key position.

[0065] At 308, a ranked collection of neighbors is determined for the data object in a neighbor graph, wherein the ranking is based on the index.

[0066] At 310, the neighbor graph is incrementally updated via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, where the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection.

[0067] At 312, an incrementally updated visual rendition of the neighbor graph is generated for the incoming data stream 102.

[0068] Figure 4 is a flow diagram illustrating one example of a method for providing an incremental update of a neighbor graph via an orthogonal transform based indexing.

[0069] At 400, an incoming data stream 102 comprising data objects is received via a processor.

[0070] At 402, an orthogonal transform is applied to a data object in the incoming data stream 102.

[0071] At 404, the data object is associated with a subset of U ordered hash positions.

[0072] At 406, an index of U ordered key positions is retrieved via the processor, where each key position is indicative of received data objects of the incoming data stream 102 that have a hash position at the key position.

[0073] At 408, a ranked collection of neighbors is determined for the data object in a neighbor graph, wherein the ranking is based on the index.

[0074] At 410, the neighbor graph is incrementally updated via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, where the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection.

[0075] At 412, an incrementally updated visual rendition of the neighbor graph is generated for the incoming data stream 102.

[0076] At 414, the incrementally updated visual rendition of the neighbor graph is provided to a computing device via the processor, where the visual rendition is provided as the incoming data stream is received.

[0077] Figure 5 is a flow diagram illustrating one example of a method for maintaining a database of ranked collections for an incremental update of a neighbor graph via an orthogonal transform based indexing.

[0078] At 500, an incoming data stream 102 comprising data objects is received via a processor.

[0079] At 502, an orthogonal transform is applied to a data object in the incoming data stream 102.

[0080] At 504, the data object is associated with a subset of U ordered hash positions.

[0081] At 506, an index of U ordered key positions is retrieved via the processor, where each key position is indicative of received data objects of the incoming data stream 102 that have a hash position at the key position.

[0082] At 508, a ranked collection of neighbors is determined for the data object in a neighbor graph, wherein the ranking is based on the index.

[0083] At 510, the neighbor graph is incrementally updated via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, where the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection.

[0084] At 512, an incrementally updated visual rendition of the neighbor graph is generated for the incoming data stream 102.

[0085] At 514, a database of ranked collections associated with data objects in the incoming data stream 102 is maintained.

[0086] Figure 6 is a flow diagram illustrating one example of a method an index for an incremental update of a neighbor graph via an orthogonal transform based indexing.

[0087] At 600, an incoming data stream 102 comprising data objects is received via a processor.

[0088] At 602, an orthogonal transform is applied to a data object in the incoming data stream 102.

[0089] At 604, the data object is associated with a subset of U ordered hash positions.

[0090] At 606, an index of U ordered key positions is retrieved via the processor, where each key position is indicative of received data objects of the incoming data stream 102 that have a hash position at the key position.

[0091] At 608, a ranked collection of neighbors is determined for the data object in a neighbor graph, wherein the ranking is based on the index.

[0092] At 610, the neighbor graph is incrementally updated via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, where the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection.

[0093] At 612, an incrementally updated visual rendition of the neighbor graph is generated for the incoming data stream 102.

[0094] At 614, the index is updated by inserting, via the processor, a unique identifier associated with the data object into each key position of the index that corresponds to a hash position for the data object.

[0095] Examples of the disclosure provide a generalized system for incremental update of a neighbor graph via an orthogonal transform based indexing. The generalized system addresses business challenges of growing costs incurred in data storage, analytic processing and data communications, when data volumes are disproportionately high compared to utility in generating valuable insight. The key proposition is to keep an approximate nearest neighbor list of size *k* for each previously received data object after the arrival of every new data object in an incoming data stream, without directly computing the similarity/distance between any pairs of data objects. This is generally achieved by maintaining an incrementally updated index, and dynamically updated ranked collections of neighbors for each data object.

[0096] Although specific examples have been illustrated and described herein, a variety of alternate and/or equivalent implementations may be substituted for the specific examples shown and described without departing from the scope of the present disclosure. This application is intended to cover any adaptations or variations of the specific examples discussed herein. Therefore, it is intended that this disclosure be limited only by the claims and the equivalents thereof.

CLAIMS

1. A system comprising:

a hash transform module to apply, via a processor, an orthogonal transform to a data object in an incoming data stream, and to associate the data object with a subset of U ordered hash positions;

an indexing module to retrieve, via the processor, an index of U ordered key positions, wherein each key position is indicative of received data objects of the incoming data stream that have a hash position at the key position;

a neighbor determination module to determine, via the processor, a ranked collection of neighbors for the data object in a neighbor graph, wherein the ranking is based on the index; and

a graph update module to incrementally update the neighbor graph via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, wherein the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection.

2. The system of claim 1, wherein the graph update module is to provide, via the processor to a computing device, an incrementally updated computer generated visual rendition of the neighbor graph for the incoming data stream.

3. The system of claim 1, wherein the neighborhood comparisons are based on comparisons between the ranked collection of neighbors for the data object and additional ranked collections of neighbors for data objects in the ranked collection.

4. The system of claim 1, wherein the indexing module is to maintain a database of ranked collections associated with data objects in the incoming data stream.

5. The system of claim 1, wherein the indexing module is to update the index by inserting, via the processor, a unique identifier associated with the data object into each key position of the index that corresponds to a hash position for the data object.

6. The system of claim 5, wherein the ranked collection of neighbors for the data object comprises top *k* nearest neighbors determined based on a count of overlapping hash positions in the updated index.

7. The system of claim 5, wherein the hash transform module is to apply a hash transform to a next data object, wherein the next data object is received after the data object, and the neighbor determination module is to determine a new ranked collection of neighbors for the next data object based on the updated index.

8. A method comprising:

receiving, via a processor, an incoming data stream comprising data objects;

applying, via the processor, an orthogonal transform to a data object in the incoming data stream;

associating, via the processor, the data object with a subset of U ordered hash positions;

retrieving, via the processor, an index of U ordered key positions, wherein each key position is indicative of received data objects of the incoming data stream that have a hash position at the key position;

determining, via the processor, a ranked collection of neighbors for the data object in a neighbor graph, wherein the ranking is based on the index;

incrementally updating the neighbor graph via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, wherein the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection; and

generating, via the processor, an incrementally updated visual rendition of the neighbor graph for the incoming data stream.

9. The method of claim 8, further comprising providing, via the processor to a computing device, the incrementally updated visual rendition of the neighbor graph, wherein the visual rendition is provided as the incoming data stream is received.

10. The method of claim 8, wherein the neighborhood comparisons are based on comparisons between the ranked collection of neighbors for the data object and additional ranked collections of neighbors for data objects in the ranked collection.

11. The method of claim 8, further comprising maintaining a database of ranked collections associated with data objects in the incoming data stream.

12. The method of claim 8, further comprising updating the index by inserting, via the processor, a unique identifier associated with the data object into each key position of the index that corresponds to a hash position for the data object.

13. The method of claim 12, wherein the ranked collection of neighbors for the data object comprises top k nearest neighbors determined based on a count of overlapping hash positions in the updated index.

14. A non-transitory computer readable medium comprising executable instructions to:

apply, via a processor, an orthogonal transform to a data object in an incoming data stream, and associating the data object with a subset of U ordered hash positions;

retrieve, via the processor, an index of U ordered key positions, wherein each key position is indicative of received data objects of the incoming data stream that have a hash position at the key position;

update the index by inserting, via the processor, a unique identifier associated with the data object into each key position of the index that corresponds to a hash position for the data object;

determine, via the processor, a ranked collection of neighbors for the data object in a neighbor graph, wherein the ranking is based on the updated index;

incrementally update the neighbor graph via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, wherein the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection; and

provide, via the processor to a computing device, an incrementally updated computer generated visual rendition of the neighbor graph for the incoming data stream.

15. The non-transitory computer readable medium of claim 14, further comprising instructions to maintain an incrementally updated database of ranked collections of neighbors for data objects in the incoming data stream.

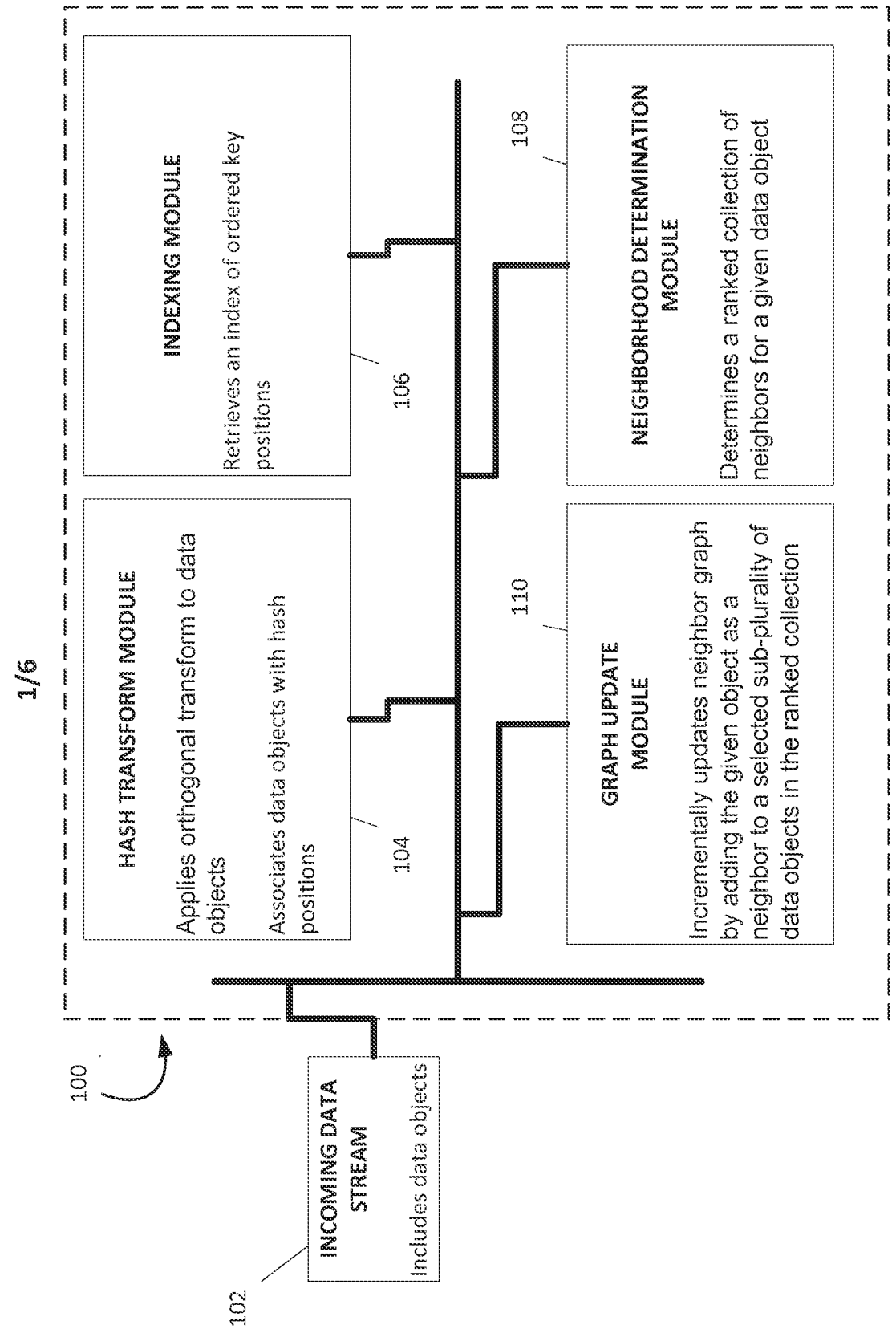

1/6
HASH TRANSFORM MODULE
Applies orthogonal transform to data objects
Associates data objects with hash positions
104
INDEXING MODULE
Retrieves an index of ordered key positions
106
GRAPH UPDATE MODULE
Incrementally updates neighbor graph by adding the given object as a neighbor to a selected sub-plurality of data objects in the ranked collection
110
NEIGHBORHOOD DETERMINATION MODULE
Determines a ranked collection of neighbors for a given data object
108
100
INCOMING DATA STREAM
Includes data objects
102

Fig. 1

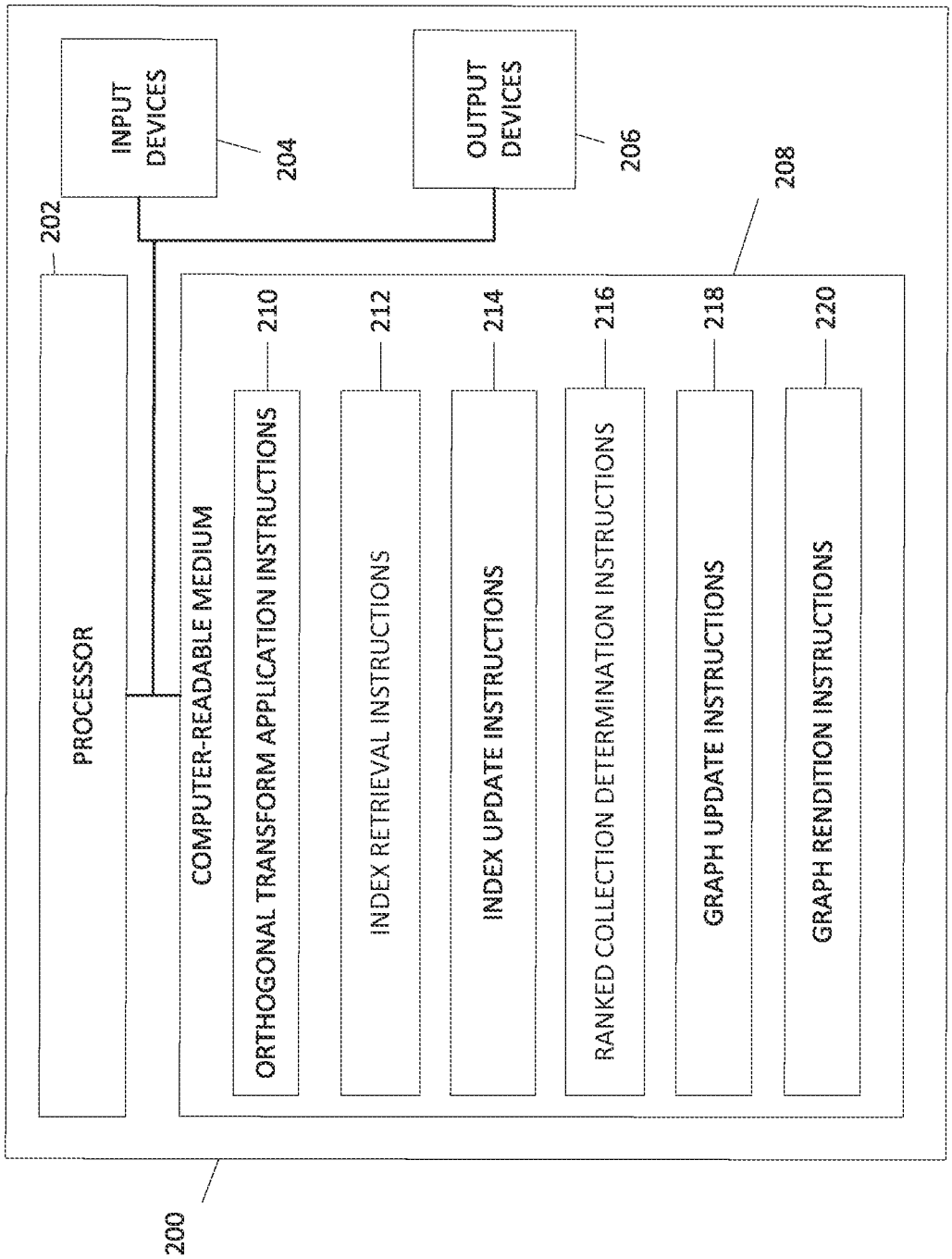

2/6
200
202
PROCESSOR
INPUT DEVICES
204
OUTPUT DEVICES
206
208
COMPUTER-READABLE MEDIUM
210
ORTHOGONAL TRANSFORM APPLICATION INSTRUCTIONS
212
INDEX RETRIEVAL INSTRUCTIONS
214
INDEX UPDATE INSTRUCTIONS
216
RANKED COLLECTION DETERMINATION INSTRUCTIONS
218
GRAPH UPDATE INSTRUCTIONS
220
GRAPH RENDITION INSTRUCTIONS

Fig. 2

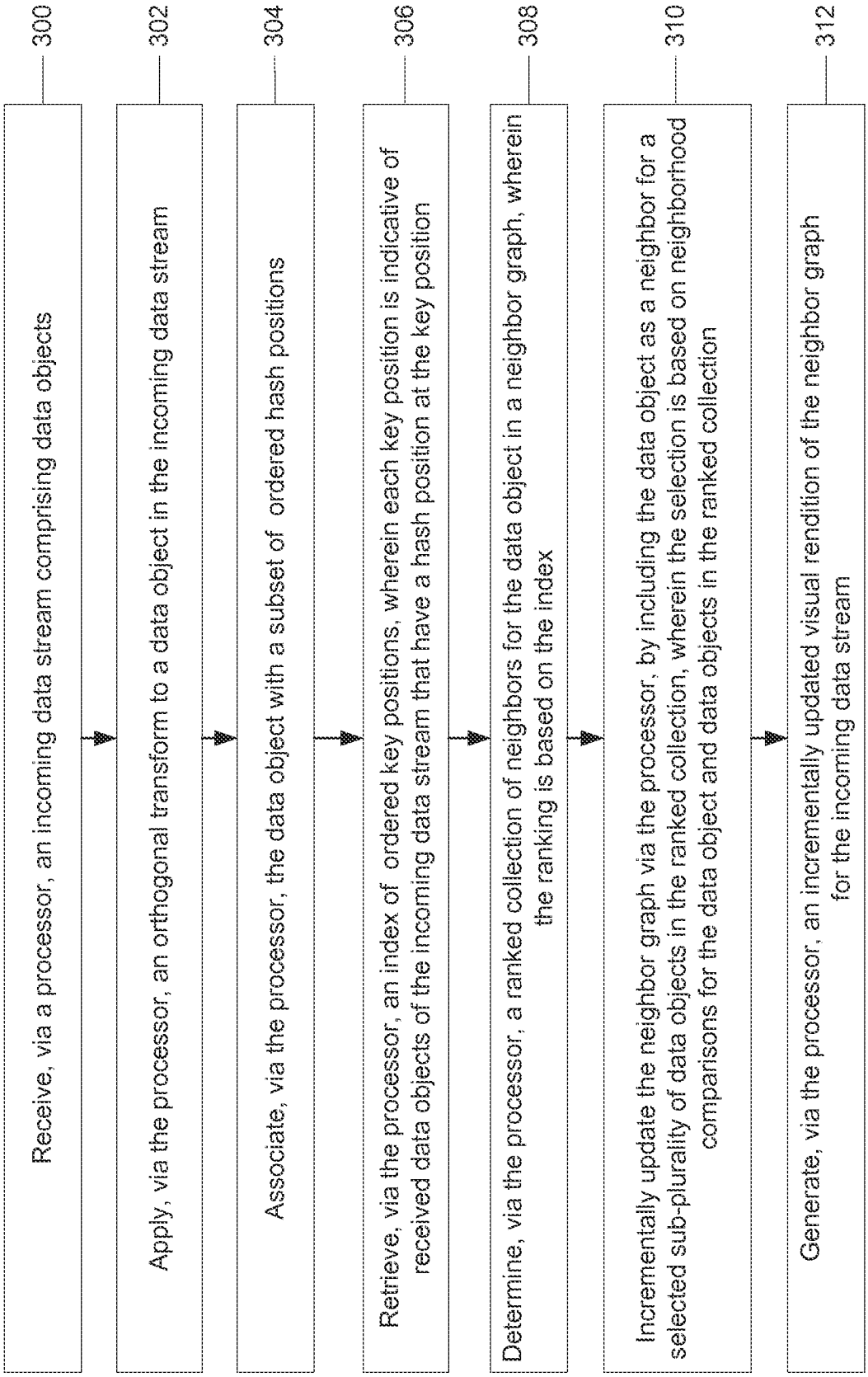

3/6
Receive, via a processor, an incoming data stream comprising data objects
300
Apply, via the processor, an orthogonal transform to a data object in the incoming data stream
302
Associate, via the processor, the data object with a subset of ordered hash positions
304
Retrieve, via the processor, an index of ordered key positions, wherein each key position is indicative of received data objects of the incoming data stream that have a hash position at the key position
306
Determine, via the processor, a ranked collection of neighbors for the data object in a neighbor graph, wherein the ranking is based on the index
308
Incrementally update the neighbor graph via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, wherein the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection
310
Generate, via the processor, an incrementally updated visual rendition of the neighbor graph for the incoming data stream
312

Fig. 3

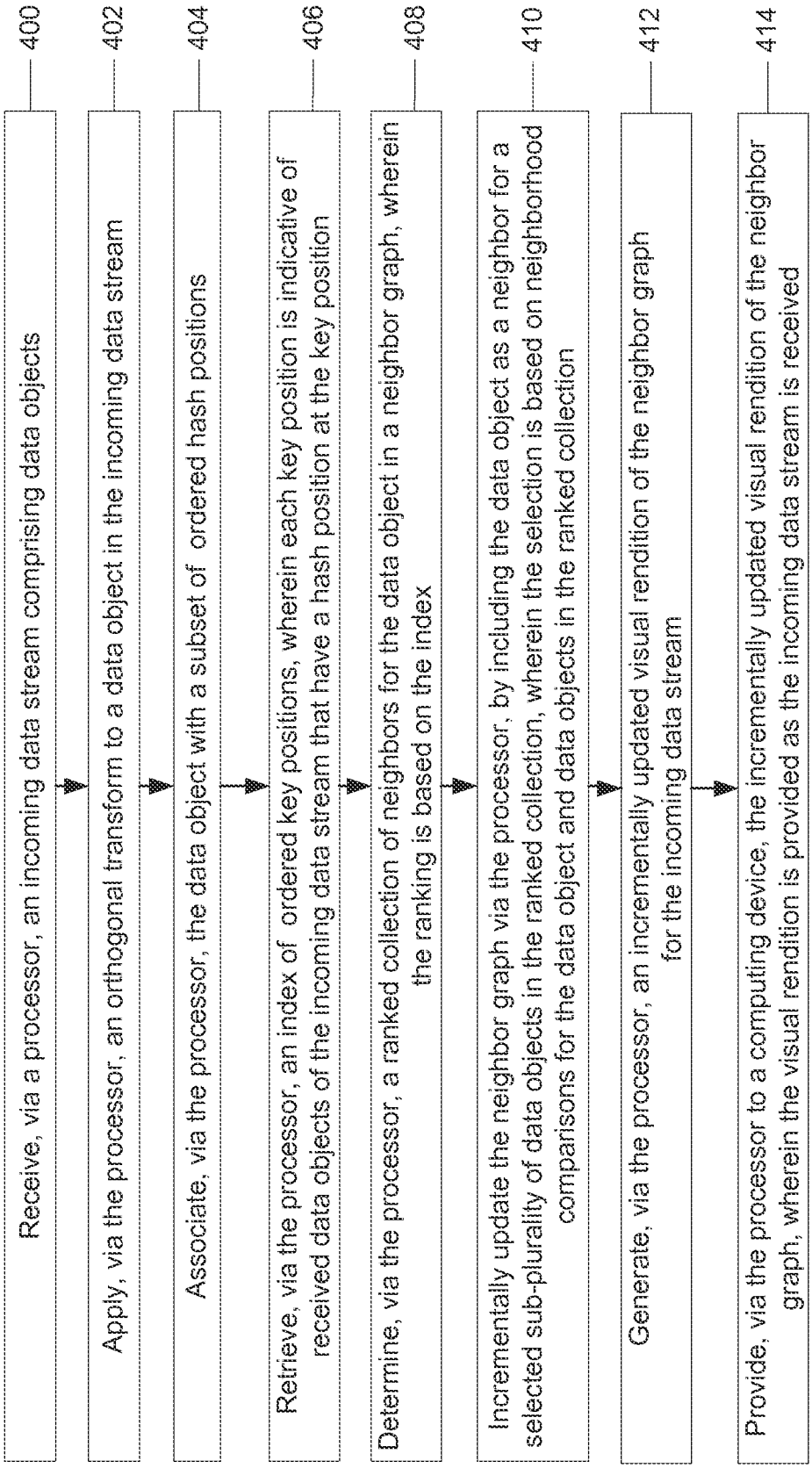

4/6
Receive, via a processor, an incoming data stream comprising data objects
400
Apply, via the processor, an orthogonal transform to a data object in the incoming data stream
402
Associate, via the processor, the data object with a subset of ordered hash positions
404
Retrieve, via the processor, an index of ordered key positions, wherein each key position is indicative of received data objects of the incoming data stream that have a hash position at the key position
406
Determine, via the processor, a ranked collection of neighbors for the data object in a neighbor graph, wherein the ranking is based on the index
408
Incrementally update the neighbor graph via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, wherein the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection
410
Generate, via the processor, an incrementally updated visual rendition of the neighbor graph for the incoming data stream
412
Provide, via the processor to a computing device, the incrementally updated visual rendition of the neighbor graph, wherein the visual rendition is provided as the incoming data stream is received
414

Fig. 4

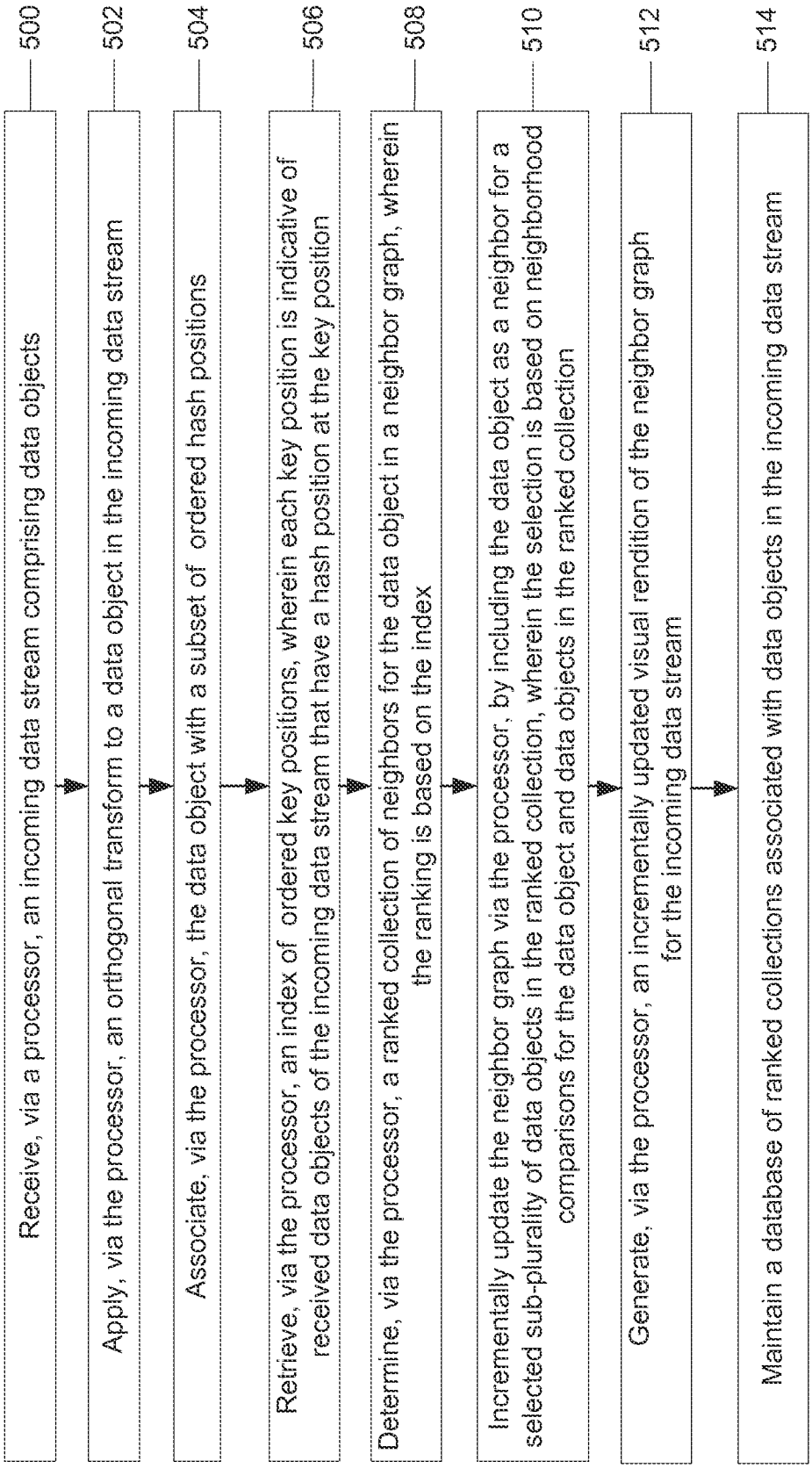

5/6
Receive, via a processor, an incoming data stream comprising data objects
500
Apply, via the processor, an orthogonal transform to a data object in the incoming data stream
502
Associate, via the processor, the data object with a subset of ordered hash positions
504
Retrieve, via the processor, an index of ordered key positions, wherein each key position is indicative of received data objects of the incoming data stream that have a hash position at the key position
506
Determine, via the processor, a ranked collection of neighbors for the data object in a neighbor graph, wherein the ranking is based on the index
508
Incrementally update the neighbor graph via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, wherein the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection
510
Generate, via the processor, an incrementally updated visual rendition of the neighbor graph for the incoming data stream
512
Maintain a database of ranked collections associated with data objects in the incoming data stream
514

Fig. 5

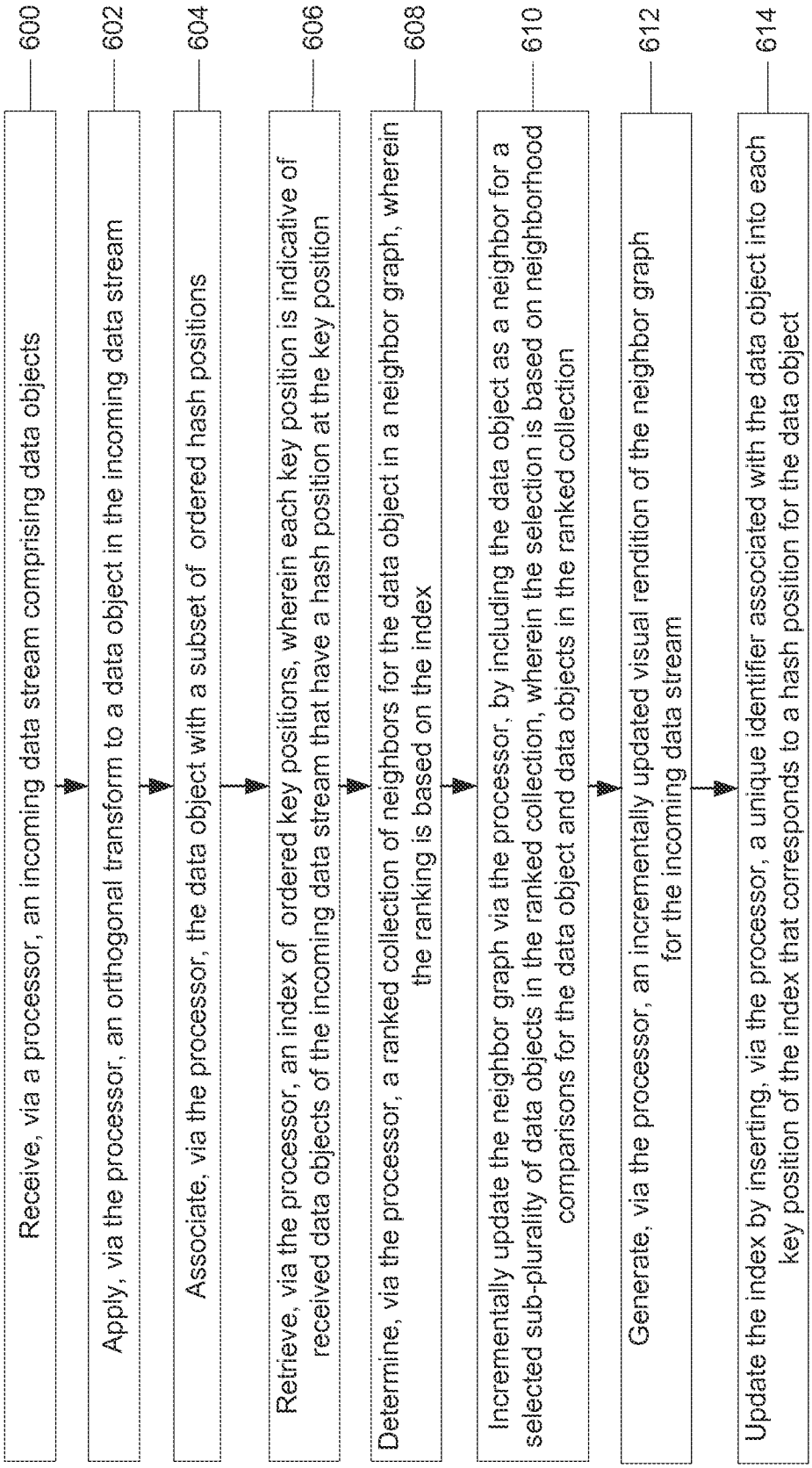

6/6
Receive, via a processor, an incoming data stream comprising data objects
600
Apply, via the processor, an orthogonal transform to a data object in the incoming data stream
602
Associate, via the processor, the data object with a subset of ordered hash positions
604
Retrieve, via the processor, an index of ordered key positions, wherein each key position is indicative of received data objects of the incoming data stream that have a hash position at the key position
606
Determine, via the processor, a ranked collection of neighbors for the data object in a neighbor graph, wherein the ranking is based on the index
608
Incrementally update the neighbor graph via the processor, by including the data object as a neighbor for a selected sub-plurality of data objects in the ranked collection, wherein the selection is based on neighborhood comparisons for the data object and data objects in the ranked collection
610
Generate, via the processor, an incrementally updated visual rendition of the neighbor graph for the incoming data stream
612
Update the index by inserting, via the processor, a unique identifier associated with the data object into each key position of the index that corresponds to a hash position for the data object
614

Fig. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/055883

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/00(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 17/00; G06F 17/10; G06N 7/00; G06F 17/30; G01C 21/34

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: hash, data object, data stream, index, neighbor, rank, update, graph, 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 2013-0151536 A1 (LEMAN AKOGLU et al.) 13 June 2013 See paragraphs [0004]-[0005], [0016]-[0019], and [0059]-[0069]; claims 1 and 19-20; and figures 1-2.	1-15
A	WO 2013-129580 A1 (OSAKA PREFECTURE UNIVERSITY PUBLIC CORPORATION) 06 September 2013 See paragraphs [0010]-[0014]; claim 1; and figure 20.	1-15
A	US 2014-0258295 A1 (MICROSOFT CORPORATION) 11 September 2014 See paragraphs [0007] and [0032]-[0047]; and figures 2-3.	1-15
A	US 2014-0046593 A1 (UGUR DEMIRYUREK et al.) 13 February 2014 See paragraphs [0005]-[0023] and figures 6-7.	1-15
A	US 2008-0033897 A1 (KENNETH A. LLOYD) 07 February 2008 See paragraphs [0021]-[0026] and figures 3-5.	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	Date of mailing of the international search report
15 July 2016 (15.07.2016)	**15 July 2016 (15.07.2016)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea
Facsimile No. +82-42-481-8578

Authorized officer
NHO, Ji Myong
Telephone No. +82-42-481-8528

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2015/055883

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2013-0151536 A1	13/06/2013	US 8903824 B2	02/12/2014
WO 2013-129580 A1	06/09/2013	None	
US 2014-0258295 A1	11/09/2014	None	
US 2014-0046593 A1	13/02/2014	US 8566030 B1 US 9062985 B2	22/10/2013 23/06/2015
US 2008-0033897 A1	07/02/2008	WO 2008-017044 A1	07/02/2008