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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 January 2017 (26.01.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/014744 A1

(51) International Patent Classification:
G06F 17/30 (2006.01) *G06F 17/00* (2006.01)

(21) International Application Number:
PCT/US2015/041150

(22) International Filing Date:
20 July 2015 (20.07.2015)

(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: **NOR, Igor**; Kiryat Technion Technion, 32000 Haifa (IL). **BARKOL, Omer**; Kiryat Technion Technion, 32000 Haifa (IL).

(74) Agents: **KIRCHEV, Ivan T.** 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, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: PROCESSING TIME-VARYING DATA USING A GRAPH DATA STRUCTURE

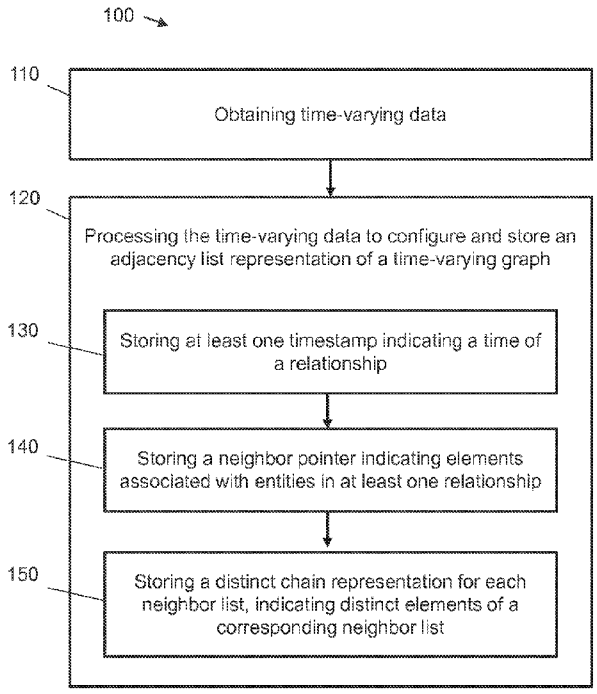

Fig. 1

(57) Abstract: Examples of processing time-varying data using a graph data structure are described. In one case, an adjacency list representation of a time-varying graph is used. In this case, vertices in the adjacency list representation correspond to defined entities in the time-varying data and each vertex has a corresponding neighbor list. Neighbor pointers are provided indicating a relationship between elements in the set of neighbor arrays. A distinct chain representation is also used for each neighbor list, wherein each distinct chain representation indicating distinct elements of a corresponding neighbor list that are ordered based on a relationship time from the time-varying data.

WO 2017/014744 A1

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))*

PROCESSING TIME-VARYING DATA USING A GRAPH DATA STRUCTURE

BACKGROUND

[0001] Many situations arise in which time-varying data is produced. For example, social media feeds, system logs, telecommunications systems and network monitoring applications may all generate records of events that occur over time.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] Various features of the present disclosure will be apparent from the detailed description which follows, taken in conjunction with the accompanying drawings, which together illustrate, features of certain examples, and wherein:

[0003] Figure 1 is a flow diagram showing a method for obtaining and processing time-varying data according to an example;

[0004] Figures 2A and 2B are schematic illustrations showing representations of relationships between elements in a time-varying graph;

[0005] Figure 3A-3C are schematic illustrations showing an adjacency list representation of a time-varying graph according to an example;

[0006] Figure 4 is a schematic illustration of an apparatus for loading a temporal portion of data according to an example; and

[0007] Figure 5 is a schematic illustration showing an adjacency list representation of a time varying graph, stored on a non-transitory computer readable medium according to an example.

DETAILED DESCRIPTION

[0008] In a social media feed a user may add or remove other users. In a telecommunications system, connectivity between a mobile device and a base station may vary over time. Similarly, repositories of electronic documents, such as the HyperText Markup Language (HTML) documents forming the World Wide Web, change over time as new documents get added and old documents are removed or modified. It is frequently desirable to quickly and efficiently store and/or retrieve such data.

[0009] It is frequently desirable to analyze time-varying data, which may be received as streaming data (i.e. a "data stream"). For example, the data may be indicative of connections at given times between entities in a network such as a computer network or a telecommunications system. As another example, the data may be indicative of e-commerce transactions wherein connections between users represent transactions between those users. In certain examples described herein this time-varying or "dynamic" data is represented as a graph in a graph data structure. This then allows the data to be analyzed using graph processing techniques.

[0010] When analyzing time-varying data, it may be desirable to analyze an up-to-date representation of the data and/or a representation of the data at a given time, or over a given time range, in the past. This time-based querying presents challenges for data storage and/or retrieval. Processing an incoming or new event in a data stream may take a given amount of time and computing resources and this may vary depending on the processing methods that are being used. In one case, a complete copy of a graph data structure could be stored every time a change in the data stream occurs. This would take considerable time and resources at the time of storage but would aid rapid retrieval of data. In certain cases it may not be possible to appropriately handle changes if they are received too frequently and/or the data structure becomes too large. In another case, changes may be stored in a log file. This would enable rapid storage of data. However, in this case, a new graph data structure would need to be constructed and processed with each data retrieval operation. As such, time-based queries may take a considerable amount of time to perform and/or fail to complete for large datasets and constrained resources. Given this, there is a desire for a time- and resource-efficient methods for storing and loading time-varying data, in particular where that data may be represented as graph data.

[0011] Certain examples described herein allow for useful processing of time-varying data. Certain examples obtain time-varying data, said data indicating at least one relationship between two defined entities that occurs at a given time. This may, for example, represent a network connection or transaction between two computing devices, or a recorded grouping or coupling between user accounts. The time-varying data is then processed to configure and store an adjacency list

representation of a time-varying graph based on the time-varying data. Vertices in the adjacency list representation correspond to defined entities in the time-varying data and each vertex has a corresponding neighbor list. The processing comprises storing, in association with the adjacency list representation, at least one timestamp indicating a respective time of the at least one relationship. A neighbor pointer is also stored, indicating elements associated with the defined entities in the at least one relationship in respective neighbor lists. For each neighbor list, a distinct chain representation is stored, each distinct chain representation indicating distinct elements of a corresponding neighbor list that are ordered based on a relationship time from the time-varying data.

[0012] Storing time-varying data as described in examples herein takes less time and uses fewer resources than storing, for each change (such as a new connection), a complete representation of the data at a given time. However, it also allows for fast and efficient loading of the data, for example to return a representation of the data at a request point or interval in time, as compared to a log-based approach.

[0013] To retrieval or load time-varying data, in certain examples described herein, a time interval is obtained associated with a temporal portion of a time-varying graph, e.g. the graph as described above. The adjacency list representation of the time-varying graph may then be loaded. This may comprise searching vertices of the adjacency list representation to locate a vertex with an associated relationship time within the requested time interval. For the located vertex, a last element is then determined within the neighbor list for the located vertex that has an associated relationship time within the time interval. This may represent the first or last connection of the vertex in question within the time interval.

[0014] Elements in the adjacency list representation may then be retrieved, starting from the determined last element. In an example, this retrieving comprises traversing, based on at the neighbor pointers, elements within the neighbor lists that have an associated relationship time within the time interval and that form part of one of the distinct chain representations. The searching of vertices and determining a last element within the neighbor list for the located vertex may comprise a binary search, and the retrieving of elements may comprise a breadth-

first search. In this manner, a representation of the graph data corresponding to the time interval may be retrieved faster and more efficiently than in a system in which the stored data comprises merely a log of each change.

[0015] As noted above, certain examples described herein provide for processing of graph data so that it may be both stored and subsequently retrieved more efficiently, in terms of time and/or computing resources, than with comparative techniques.

[0016] Figure 1 shows a method 100 for processing time-varying data according to an example. In this case, the time-varying data, obtained at block 110, indicates at least one relationship between two defined entities that occurs at a given time. Time-varying data may relate to an event stream, e.g. a series of events wherein each event involves two or more entities and occurs at a defined time. The entities may be computers in a network, and the at least one relationship may indicate a connection between these computers, e.g. as extracted by a packet sniffing tool. As another example, the entities may comprise elements of a telecommunications network, such as mobile devices and base stations or core network elements, with the at least one relationship similarly representing a connection between these elements. As a further example, the entities may represent users in a computing application, e.g. as identified by a user account having an appropriate user identifier. The relationship in this case may comprise adding a particular second user to a user list of a first user. In another case, the relationship may comprise a hyperlink and the entities may comprise HTML documents on the World Wide Web. It will be clear to one skilled in the art that the presently described method is applicable to other forms of data indicating relationships between entities.

[0017] At block 120, the time-varying data is processed to configure and store an adjacency list representation of a time-varying graph, based on the time-varying data. Vertices in the adjacency list representation correspond to defined entities in the time varying data and each vertex has a corresponding neighbor list. The neighbor list indicates vertices with which connections exist to the vertex in question. In certain cases, the adjacency list representation may comprise a plurality of array data structures. For example, a first array data structure may store a list of vertices and each entry in the list of vertices may have a corresponding

linked array data structure implementing a neighbor list. These array structure may be dynamic, e.g. they may change in size over time.

[0018] In Figure 1, the processing at block 120 includes a number of sub-blocks. At sub-block 130, at least one timestamp indicating a respective time of the at least one relationship is stored. For example, the timestamp may indicate a time of connection between two computers in a network.

[0019] At sub-block 140, the processing comprises storing a neighbor pointer indicating elements associated with the defined entities in the at least one relationship in respective neighbor lists. For example, a connection may exist between vertices *A* and *B*. *B* would then be in the neighbor list of *A*, and *A* would be in the neighbor list of *B*. In such a situation, a neighbor pointer associated with the representation of *A* in the neighbor list of *B* would point to the representation of *B* in the neighbor list of *A*. For example, the neighbor pointer may comprise a tuple data structure that stores the indices of *A* and *B* in their respective neighbor lists, in certain cases together with a timestamp indicating the time of connection. In one case, the timestamp is stored in association with the neighbor pointer.

[0020] At sub-block 150, a distinct chain representation is stored for each neighbor list. The distinct chain representation indicates distinct elements of a corresponding neighbor list. As such, if a given element is repeated in a neighbor list, the distinct chain representation would indicate one instance of that element. The distinct chain representation may be implemented as a hash table.

[0021] Figures 2A and 2B show examples of relationships between elements that may be stored as described above. The relationships are shown as edges between vertices of a time-varying graph 200. The time-varying graph comprises vertices V_1 (210), V_2 (220), V_3 (230) and V_4 (240). Figure 2A relates to a first time or time period t_1. At time t_1, a connection occurs between vertex V_1 (210) and vertex V_2 (220). This is represented by edge 215. Figure 2B relates to a time or time period t_2, at which time a connection occurs between vertex V_1 (210) and vertex V_3 (230). This is represented by edge 225.

[0022] Figures 3A-3C are schematic illustrations that show an adjacency list representation of a time-varying graph according to an example. In Figure 3A, an array or vertex list 300, which may be a dynamic array, is stored in which elements

310-1 … 310-n represent vertices of the time-varying graph V_1 … V_n, including vertices V_i (310-i) and V_j (310-j).

[0023] For each vertex a corresponding array is stored representing a corresponding neighbor list (320-1 … 320-n). These neighbor lists 320 may also comprise dynamic arrays. Each neighbor list 320 comprises a list of vertex identifiers, each identifier indicating a connection between the identified vertex in the neighbor list and the corresponding vertex in the vertex list 300. For example, Figure 3A shows that a connection exists in the time-varying graph between vertex V_i and vertex V_j. This may comprise the edge 215 or 225 as shown in one of Figures 2A and 2B. As such, the neighbor list 320-i of vertex V_i includes an element 330-j corresponding to vertex V_j, and the neighbor list 320-j of vertex V_j will include an element 330-i corresponding to vertex V_i. The elements of the neighbor lists 320-1 … 320-n may be stored in an order indicating the order in which the connections occurred. For example, in Figure 3A elements to the right-hand side of the Figure may represent newer connections, wherein a far right element represents the most recent connection or graph edge.

[0024] Figure 3A also indicates schematically a neighbor pointer 340 that is stored. The neighbor pointer 340 indicates a connection between the element 330-j corresponding to vertex V_j in the neighbor list 320-i of vertex V_i, and the element 330-i corresponding to vertex V_i in the neighbor list 320-j of vertex V_j. The neighbor pointer 340 may comprise references to memory locations storing identifiers for vertices V_i and V_j. Given a known element in one of the neighbor lists 330-i and 330-j, the neighbor point 340 enables the corresponding element in the other neighbor list to be located.

[0025] Certain examples described above enable events from a data stream (time-varying data) to be efficiently stored. If data is stored as described above, efficient loading or retrieval of data is enabled. An example of retrieving a portion of data corresponding to a particular time interval will now be described.

[0026] Given an adjacency list representation, as schematically shown in Figure 3A, a retrieval operation begins by searching vertices of the vertex list 300 to locate any vertex with an associated relationship time within the time interval. This may for example be a binary search. A second search, which may also be a binary search, is then performed within the neighbor list 330 of the located vertex,

to find a vertex with an associated relationship time within the time interval. This may for example correspond to the first or last relationship in the neighbor list within the time interval. Elements of the adjacency list representation may then be retrieved by starting at the determined element and traversing, based on the neighbor pointers, elements within neighbor lists that have an associated relationship time within the time interval. This may for example be a breadth-first search. This retrieval operation retrieves vertices and edges or connections from the graph data structure that are within the queried time interval.

[0027] Figure 3B shows a representation of a neighbor list 330 for a given vertex in Figure 3A. The neighbor list 330-B comprises elements corresponding to connections from the given vertex to other vertices in the graph data structure, the other vertices being represented by the characters *a*, *b*, *c*, *d*, *e* (e.g. said characters comprise identifiers for the vertices). Multiple connections occur between the given vertex and some of the other vertices, and as such the neighbor list 330-B may comprise multiple elements corresponding to a single other vertex. For example, the repeated inclusion of the identifier for vertex a, may represent multiple connections over time between the given vertex and vertex a.

[0028] In the present described example, when retrieving graph data for a requested time or time interval, a distinct chain representation is used to determine all vertices that have connections that occur within that time interval. In this case, it is deemed sufficient to know that at least one connection occurred between two vertices in a requested time interval, without returning further information regarding the nature of any connections within the time interval. By returning information regarding entities active within a time period, but reducing the amount of information regarding specific connections within that time period, graph retrieval operations may be increased in speed. In this case, a distinct chain representation 350 is stored for each neighbor list. The distinct chain representation indicates distinct elements of its corresponding neighbor list. That is to say, it skips repeated instances of the elements corresponding to connections with the same vertex. In Figure 3B, the distinct chain representation indicates the first occurrence of a connection to a particular other vertex. In other embodiments, it may for example indicate the last occurrence of a connection to a particular other vertex.

[0029] As illustrated in Figure 3B, in one example, the distinct chain representation may comprise at least one pointer, each said pointer being associated with a distinct element of the corresponding neighbor list and referencing a further distinct element of the corresponding neighbor list. In this manner, a set of pointers may link the distinct elements of the neighbor list. As another example, the distinct chain representation may comprise a hash table indicating distinct elements of the corresponding neighbor list. The distinct chain representation may comprise at least one timestamp associated with an element of the distinct chain representation. For example, where the distinct chain representation comprises at least one pointer associated with a distinct element of the corresponding neighbor list, a timestamp may be associated with each of these pointers indicating the time at which that pointer was created. Figure 3C shows the array structure of Figure 3A, wherein a distinct chain representation 350-1 … 350-n is stored for each neighbor list 320-l … 320-j.

[0030] In certain examples, the use of a distinct chain representation may be extended to include additional information regarding connections within a requested time or time period. In this case, at least one count indicating a number of relationships between at least two defined entities occurring in a given time interval may be stored in association with the adjacency list representation. In one example, this count may be determined by storing a succinct data structure for each neighbor list. In this case, a succinct data structure is a data structure that uses an amount a space within a defined threshold of the information-theoretic lower bound. Unlike certain lossless data compression algorithms, succinct data structures may be used “in-place” without decompression. A succinct data structure may be based on a succinct indexable dictionary and enable “rank” and “select” methods. The at least one count may then be retrieved by comparing, using the succinct data structure of a first element associated with one of the two defined entities, a rank of a second element associated with another of the two defined entities at the start and end of the given time interval. Typically such a rank function *Rank(value, location)* indicates the number of times that the given *value* appears in an array from the beginning up to the given *location*. For example, it may be desirable to calculate the number of times the vertex V_j appears in the neighbor list of vertex V_i, within the time interval (t_1, t_2). This may be achieved by,

for V_i, subtracting *Rank*(V_j, t_1) from *Rank*(V_j, t_2). This provides the number of relationships between vertices V_i and V_j in the time interval (t_1, t_2).

[0031] Returning to Figure 3C, when an indication of a new connection between vertices V_i and V_j is obtained, for example as part of streaming data, an element corresponding to V_i is stored at one end of the neighbor list 320-j corresponding to V_j, and an element corresponding to V_j is stored at one end of the neighbor list 320-i corresponding to V_i. A neighbor pointer is then stored indicating the connection between these elements. The distinct chain representation is also updated. For example, as described above, the distinct chain representation may identify the most recent connection between a vertex and each relevant other vertex. On obtaining a new connection between vertices V_i and V_j, The distinct chain representation is updated to identify that connection instead of any previous connection between vertices V_i and V_j. A timestamp may also be stored, indicative of the time of updating the distinct chain representation. As such, for each new element to be added to a neighbor list, storing a distinct chain representation for said neighbor list comprises adding the new element to an end of the distinct chain representation and removing any element having a same value as the new element from the distinct chain representation. For example, starting from Figure 3B, a new connection to element *c* would result in the links from *b* to *c* and *c* to *d* being removed and replaced with a new link from *b* to *d* (in addition to a new link from *c* to *a*). This process takes a constant period of time (O(1)) for each new connection and so is independent of the size of the dataset. It is thus straightforward to assign computing resources for the storage of new connections.

[0032] In some embodiments, representations of relationships may be removed from the adjacency list representation. For example, relationships may be removed after a given amount of time has elapsed. Removing a relationship between vertices V_i and V_j comprises removing elements corresponding to the relationship from the neighbor list 320-i of vertex V_i and from the neighbor list 320-j of vertex V_j. A timestamp may be stored indicating the time at which the removal occurred. As the elements of each neighbor list are time-ordered, removal of the earliest stored relationship comprises removing the first element of the corresponding neighbor lists.

[0033] Certain examples described above enable events from a data stream, (time-varying data) to be efficiently stored. In the following passages examples of a corresponding loading or data retrieval operation are described. These examples enable portions of a time-varying graph data structure to be retrieved in response to a query having a particular time or time period.

[0034] In an example data retrieval operation, a time interval associated with a temporal portion of a time-varying graph is first obtained. A representation of the time-varying graph data may then be loaded that corresponds to that time interval. In this example, vertices of the above-described adjacency list representation are searched to locate a vertex with an associated relationship time within the time interval. This may be, for example, a binary search. For the located vertex, a search, which again may be a binary search, is performed within its neighbor list to determine an element that has an associated relationship time within the time interval. This may correspond to the first relationship in the neighbor list within the time interval, in examples in which the distinct chain representation indicates the first occurrence of each vertex, such that the element is represented in the distinct chain representation. Similarly, in examples in which the distinct chain representation indicates the last occurrence of each vertex, this may advantageously correspond to the last relationship in the neighbor list within the time interval.

[0035] Elements in the adjacency list representation may then be retrieved by starting at the determined element and traversing, based on the neighbor pointers, elements within neighbor lists that have an associated relationship time within the time interval and that form part of one of the distinct chain representations. This may for example be a breadth-first search. Upon following a neighbor pointer, this may comprise advancing through the distinct chain until the last element of the time interval, then travelling backward until reaching the end of the time interval. In this manner, a representation of the relationships associated with the time period may be efficiently loaded.

[0036] If a binary searches are used these may take $O(log(E))$ time to perform, where E is the number of edges or connections present in the graph data structure. Without a distinct chain representation, a breadth first search may take $O(V_I + E_I)$ time to perform, where V_I represents the number of vertices or entities within the

requested time interval and E_I represents the number of edges or connections within the requested time interval. In this case, a complexity of the loading operation is $O(log(E)+ V_I + E_I)$. When a distinct chain representation is used, a breadth first search may take $O(V_I^* + E_I^*)$ time to perform, where V_I^* represents the number of unique or distinct vertices or entities within the requested time interval and E_I^* represents the number of unique or distinct edges or connections within the requested time interval. In this case, a complexity of the loading operation is $O(log(E)+ V_I^* + E_I^*)$. In both cases, the time required may be modelled and appropriate computing resources may be assigned to perform the operation.

[0037] Figure 4 shows a schematic illustration of an apparatus for loading a temporal portion of data representing a time varying graph according to an example 400. The apparatus comprises a data interface 410 to obtain a time interval associated with the temporal portion, and a graph search processor 420 to load an adjacency list representation of the time varying data, for example stored as described above. In the example of Figure 4 the graph search processor 420 comprises a number of modules 430. These include a vertex list search module 440, a neighbor list search module 450 and a graph traversal module 460.

[0038] The vertex list search module 440 is configured to search a vertex list of the adjacency list representation to locate a vertex with an associated relationship time within the time interval. The search may for example be a binary search. The vertex list search module 440 is configured to pass the located vertex to the neighbor list search module 450. The neighbor list search module 450 determines a last element within a neighbor list for the located vertex that has an associated relationship time within the time interval. This may be performed using a binary search.

[0039] Following the operations of the vertex list search module 440 and the neighbor list search module 450, the graph traversal module 460 is configured to retrieve elements in the adjacency list representation starting from the last element determined by the neighbor list search module 450. The graph traversal module 460 is configured to traverse elements within neighbor lists of the adjacency list representation that have an associated relationship time within the time interval and that form part of a distinct chain representation for example as described

above. The graph traversal module 460 may be configured to perform the traversing as a breadth-first traversal.

[0040] The traversing is based on at least one neighbor pointer indicating connected elements in the graph from respective neighbor lists, each distinct chain representation indicating distinct elements of a corresponding neighbor list that are ordered based on a relationship time. The graph traversal module 460 may be configured to traverse new elements by advancing through the distinct chain representation until a last element in the time interval is located and then by traversing the distinct chain representation in a reverse direction until an end of the time interval is reached. The traversing may comprise traversing at least one neighbor list corresponding to an element referenced by the at least one neighbor pointer.

[0041] The apparatus 400 may further comprise a data storage device (not shown in Figure 4) configured to store the adjacency list representation. Figure 5 shows an example computer-readable medium 510 that may implement the data storage device. In such examples, the processor may be configured to receive data indicating at least one relationship between two defined entities that occurs at a given time and to store, as part of the adjacency list representation, at least one timestamp indicating a respective time of the at least one relationship. The processor may also be configured to store, as part of the adjacency list representation, a neighbor pointer indicating elements associated with the defined entities in the at least one relationship in respective neighbor lists, and to update, as part of the adjacency list representation, distinct chain representations for each neighbor list corresponding to the defined entities.

[0042] Figure 5 depicts a non-transitory computer-readable medium 510 storing data 520 representative of a dynamic graph. The computer-readable medium 510 may be connected to a processor 530. The computer-readable medium 510 may comprise any machine-readable storage media, e.g. such as a memory and/or a storage device. Machine-readable storage media can comprise any one of many physical media such as, for example, electronic, magnetic, optical, electromagnetic, or semiconductor media. More specific examples of suitable machine-readable media include, but are not limited to, a hard drive, a random access memory (RAM), a read-only memory (ROM), an erasable

programmable read-only memory, or a portable disc. In one case, the processor 530 may be arranged to store data 520 in memory such as RAM during active storing and/or loading of graph data. In another example, processor 530 may be arranged to store data 520 in a persistent storage device such as a hard disk or solid state device as part of a storing and/or loading operation.

[0043] Turning to Figure 5, the data 520 comprises an adjacency data structure 540, the adjacency data structure comprising an array of vertices 545 and a set of neighbor pointer arrays 548 corresponding to said vertices, for example as described above. The data 520 further comprises data 550 representative of neighbor pointers associated with the adjacency data structure. Each neighbor pointer indicates a graph relationship between elements in the set of neighbor arrays, each neighbor pointer comprising a timestamp indicating a graph relationship time. In addition, the data 520 comprises data 560 representative of a distinct chain representation for each neighbor array, each distinct chain representation indicating distinct elements of a corresponding neighbor array, the distinct elements being ordered based on corresponding graph relationship times.

[0044] Hence, the data 520 may comprise the data that is generated as part of the store operation described above. It may also comprise data that is interrogated to load a portion of a time-varying graph as described above.

[0045] Certain examples described above allow for efficient storing and subsequent loading of time-varying graph data, for example received as streaming data. As explained above, storing a log file of received events allows for rapid storage and requires relatively low disk space. However, retrieving data involves constructing a new graph data structure for each retrieval operation, which may take a considerable amount of time to perform and/or fail to complete for large datasets and constrained resources. Conversely, storing a complete copy of a graph data structure for each received event allows for efficient retrieval but requires significant time and resources at the time of storage. Certain examples described herein allow for such data to be efficiently received, stored, and subsequently loaded. This allows for such processing and storage to be implemented with a reduced time and computing resource burdens.

[0046] The preceding description has been presented to illustrate and describe examples of the principles described. This description is not intended to

be exhaustive or to limit these principles to any precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is to be understood that any feature described in relation to any one example may be used alone, or in combination with other features described, and may also be used in combination with any features of any other of the examples, or any combination of any other of the examples.

What is claimed is:

1. A method for processing time-varying data, comprising:

obtaining time-varying data, said data indicating at least one relationship between two defined entities that occurs at a given time; and

processing the time-varying data to configure and store an adjacency list representation of a time-varying graph based on the time-varying data,

wherein vertices in the adjacency list representation correspond to defined entities in the time-varying data and each vertex has a corresponding neighbor list,

wherein said processing comprises:

storing, in association with the adjacency list representation, at least one timestamp indicating a respective time of the at least one relationship;

storing a neighbor pointer indicating elements associated with the defined entities in the at least one relationship in respective neighbor lists; and

storing a distinct chain representation for each neighbor list, each distinct chain representation indicating distinct elements of a corresponding neighbor list that are ordered based on a relationship time from the time-varying data.

2. The method of claim 1, wherein the distinct chain representation comprises at least one pointer, each said pointer being associated with a distinct element of the corresponding neighbor list and referencing a further distinct element of the corresponding neighbor list.

3. The method of claim 1, wherein the distinct chain representation comprises a hash table indicating distinct elements of the corresponding neighbor list.

4. The method of claim 1, comprising:

obtaining a time interval associated with a temporal portion of the time-varying graph;

loading the adjacency list representation of the time-varying graph, including:

searching vertices of the adjacency list representation to locate a vertex with an associated relationship time within the time interval;

for the located vertex, determining a last element within the neighbor list for the located vertex that has an associated relationship time within the time interval; and

retrieving elements in the adjacency list representation starting from the determined last element, said retrieving comprising traversing, based on at the neighbor pointers, elements within the neighbor lists that have an associated relationship time within the time interval and that form part of one of the distinct chain representations.

5. The method of claim 4, wherein said searching vertices and determining a last element within the neighbor list for the located vertex comprises a binary search and said retrieving elements comprises a breadth-first search.

6. The method of claim 1, wherein, for each new element to be added to a neighbor list, storing a distinct chain representation for said neighbor list comprises:

adding the new element to an end of the distinct chain representation; and

removing any element having a same value as the new element from the distinct chain representation.

7. The method of claim 1, comprising storing, in association with the adjacency list representation, at least one count indicating a number of relationships between the at least two defined entities occurring in a given time interval.

8. The method of claim 7, comprising storing a succinct data structure for each neighbor list,

wherein the at least one count is retrieved by comparing, using the succinct data structure of a first element associated with one of the two defined entities, a

rank of a second element associated with another of the two defined entities at the start and end of the given time interval.

9. An apparatus for loading a temporal portion of data representing a time-varying graph comprising:

a data interface to obtain a time interval associated with the temporal portion;

a graph search processor to load an adjacency list representation of the time-varying graph, the graph search processor comprising:

a vertex list search module to search a vertex list of the adjacency list representation to locate a vertex with an associated relationship time within the time interval;

a neighbor list search module to, for the located vertex, determine a last element within a neighbor list for the located vertex that has an associated relationship time within the time interval; and

a graph traversal module to retrieve elements in the adjacency list representation starting from the determined last element, the graph traversal module being configured to retrieve elements by traversing elements within neighbor lists of the adjacency list representation that have an associated relationship time within the time interval and that form part of a distinct chain representation, said traversing being based on at least one neighbor pointer indicating connected elements in the graph from respective neighbor lists, each distinct chain representation indicating distinct elements of a corresponding neighbor list that are ordered based on a relationship time.

10. The apparatus of claim 9, wherein the vertex list search module is configured to search the vertex list of the adjacency list representation by performing a binary search.

11. The apparatus of claim 9, wherein the graph traversal module is configured to traverse elements within neighbor lists by performing a breadth-first search traversal.

12. The apparatus of claim 9, wherein the graph traversal module is configured to traverse new elements by advancing through the distinct chain representation until a last element in the time interval is located and then by traversing the distinct chain representation in a reverse direction until an end of the time interval is reached.

13. The apparatus of claim 9, wherein the graph traversal module is configured to traverse elements within neighbor lists by traversing at least one neighbor list corresponding to an element referenced by the at least one neighbor pointer.

14. The apparatus of claim 9, comprising:

a data storage device configured to store the adjacency list representation, wherein the graph search processor is configured to:

receive data indicating at least one relationship between two defined entities that occurs at a given time;

store, as part of the adjacency list representation, at least one timestamp indicating a respective time of the at least one relationship;

store, as part of the adjacency list representation, a neighbor pointer indicating elements associated with the defined entities in the at least one relationship in respective neighbor lists; and

update, as part of the adjacency list representation, distinct chain representations for each neighbor list corresponding to the defined entities.

15. A non-transitory computer-readable medium storing data representative of a dynamic graph, the data comprising:

an adjacency data structure, the adjacency data structure comprising an array of vertices and a set of neighbor arrays corresponding to said vertices;

data representative of neighbor pointers associated with the adjacency data structure, each neighbor pointer indicating a graph relationship between elements in the set of neighbor arrays, each neighbor pointer comprising a timestamp indicating a graph relationship time; and

data representative of a distinct chain representation for each neighbor array, each distinct chain representation indicating distinct elements of a corresponding neighbor array, the distinct elements being ordered based on corresponding graph relationship times.

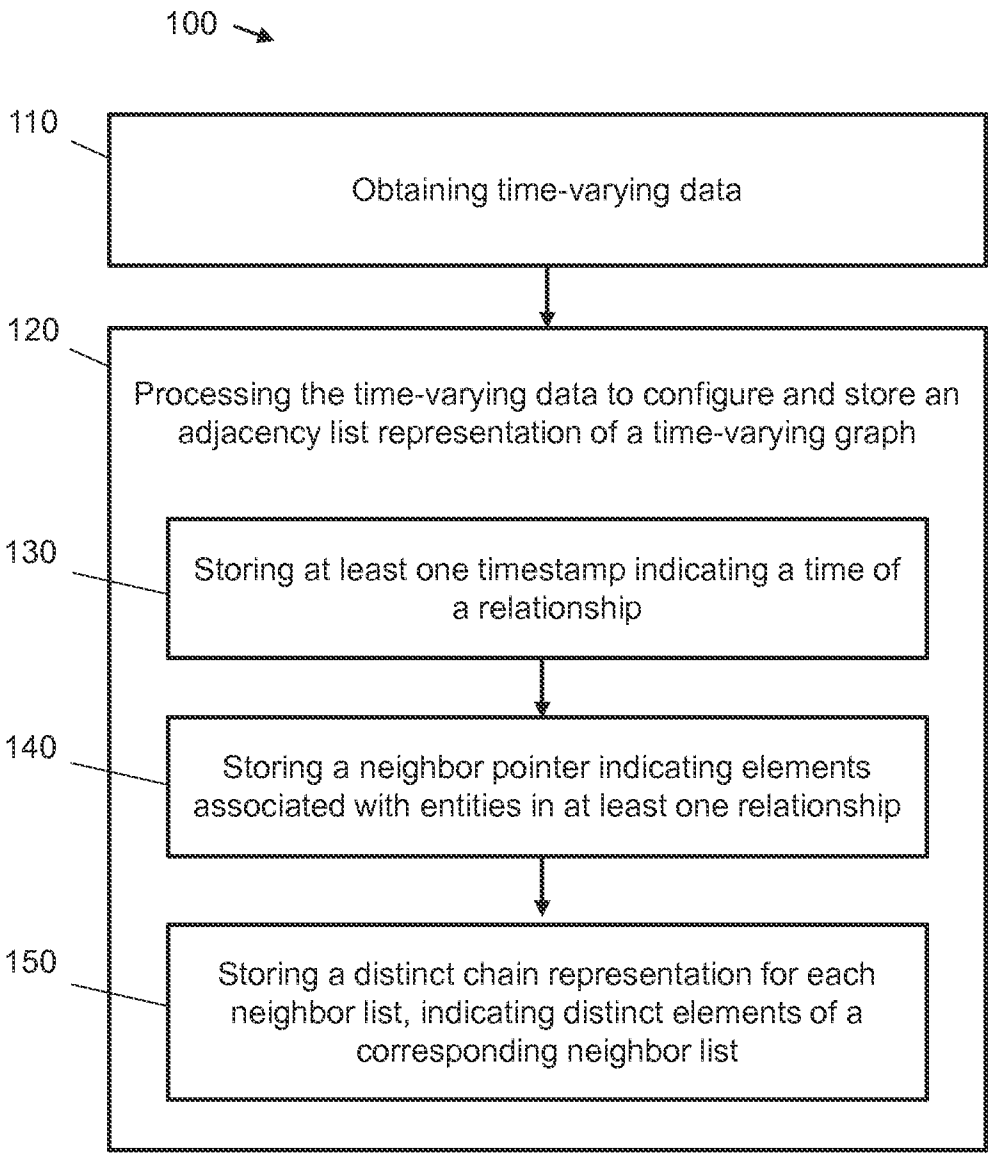
100
110
Obtaining time-varying data
120
Processing the time-varying data to configure and store an adjacency list representation of a time-varying graph
130
Storing at least one timestamp indicating a time of a relationship
140
Storing a neighbor pointer indicating elements associated with entities in at least one relationship
150
Storing a distinct chain representation for each neighbor list, indicating distinct elements of a corresponding neighbor list

Fig. 1

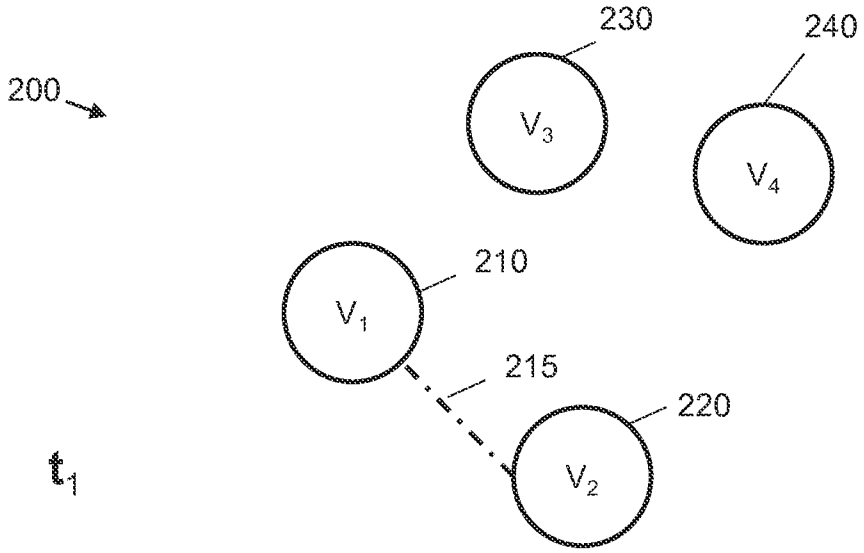
200
230
240
V_3
V_4
210
V_1
215
220
V_2
t_1

Fig. 2A

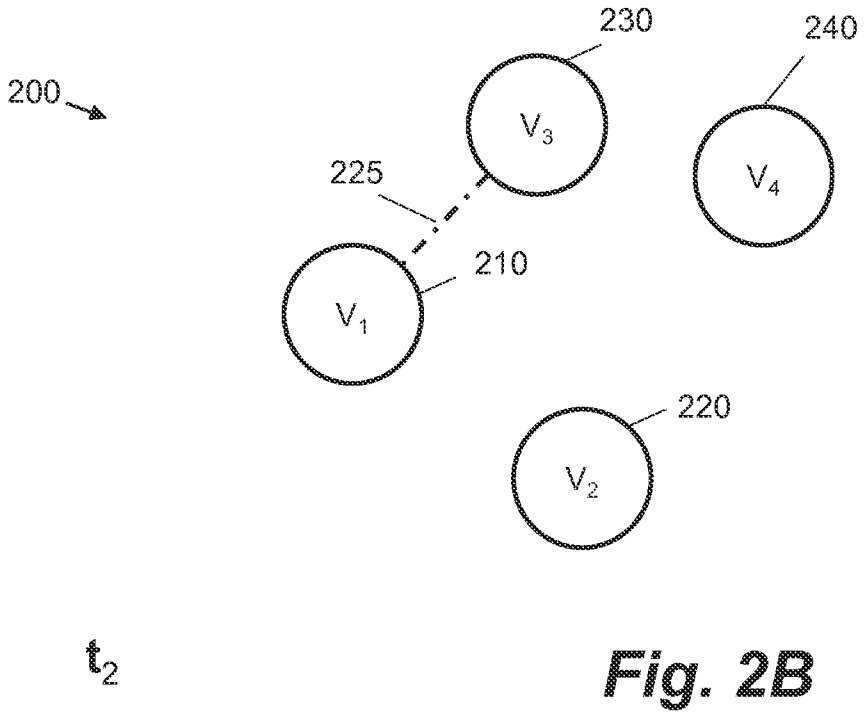
200
230
240
V_3
V_4
225
210
V_1
220
V_2
t_2

Fig. 2B

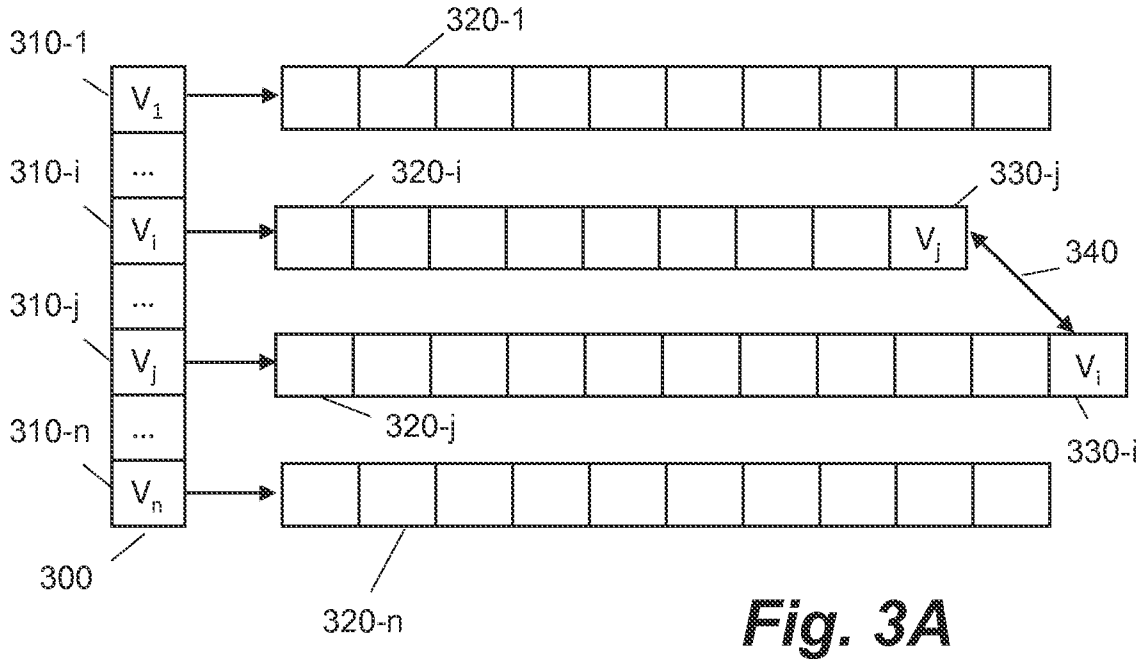

Fig. 3A

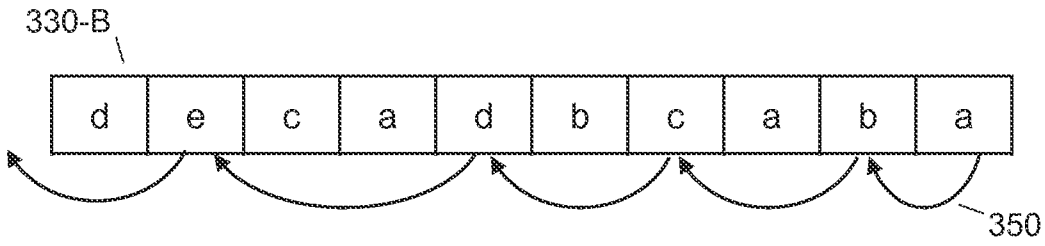

Fig. 3B

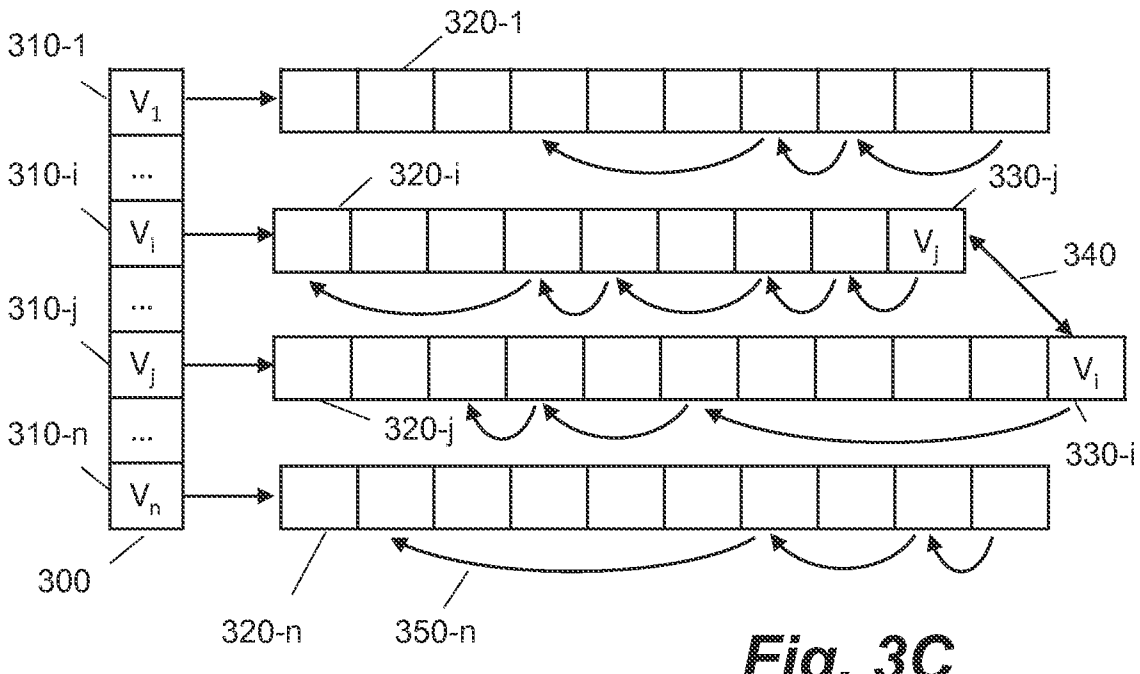

Fig. 3C

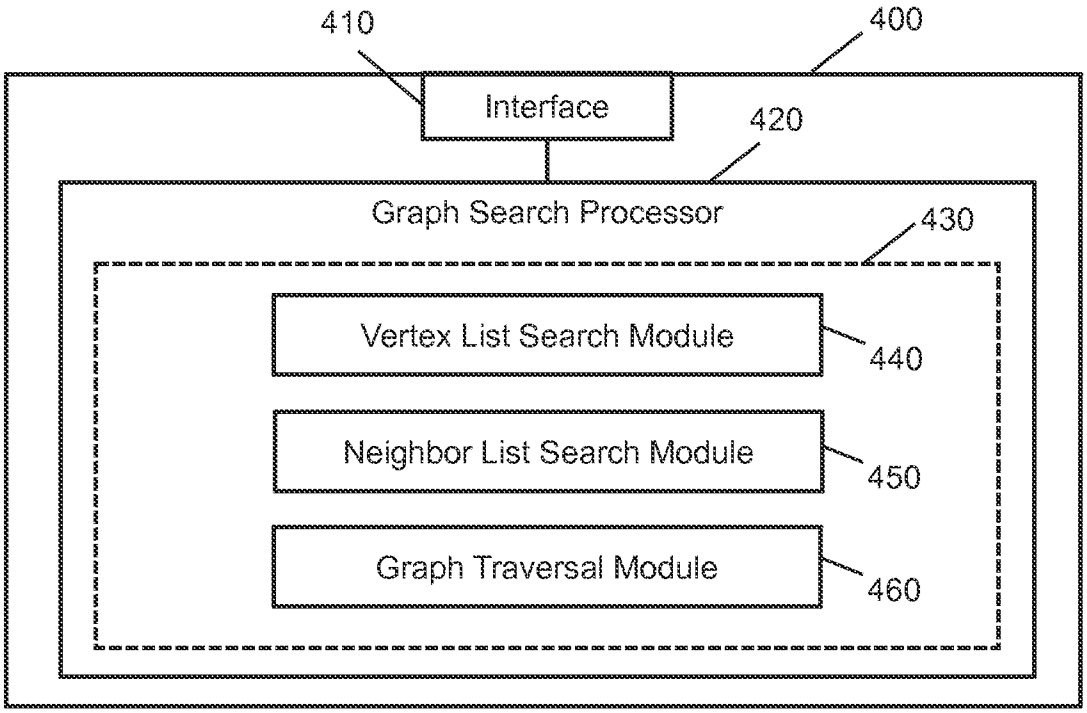
410
400
Interface
420
Graph Search Processor
430
Vertex List Search Module
440
Neighbor List Search Module
450
Graph Traversal Module
460

Fig. 4

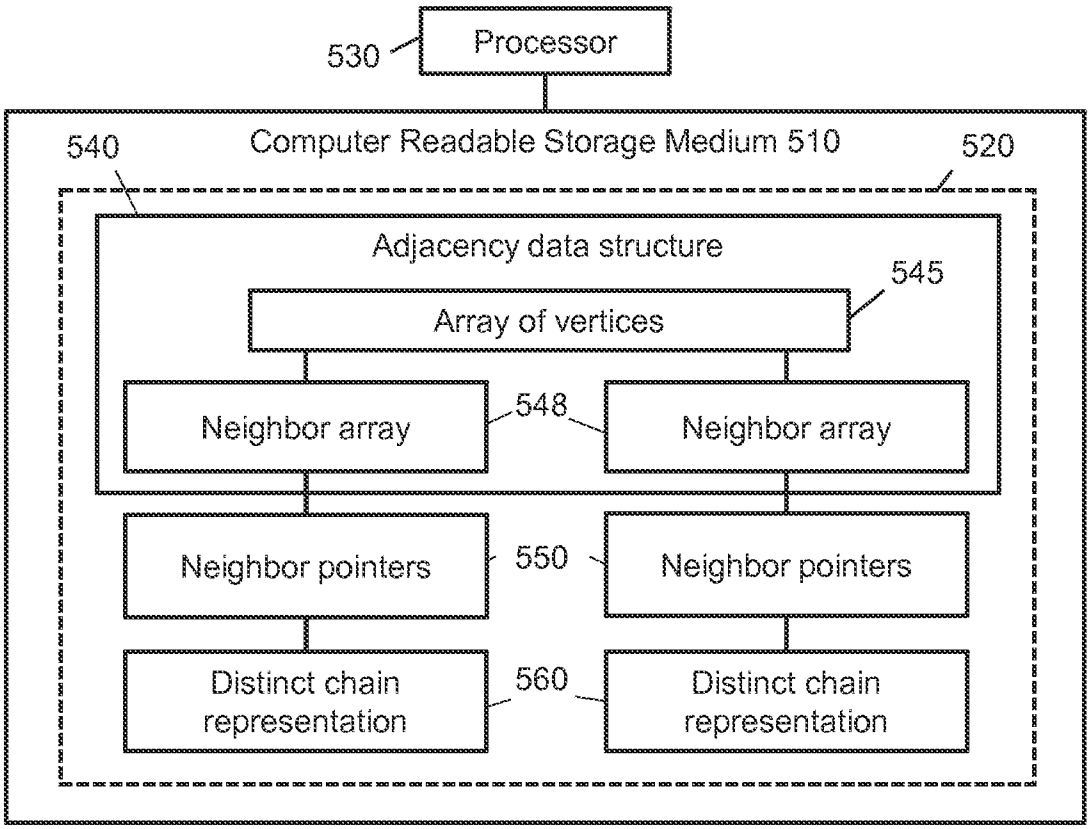
530
Processor
540
Computer Readable Storage Medium 510
520
Adjacency data structure
545
Array of vertices
Neighbor array
548
Neighbor array
Neighbor pointers
550
Neighbor pointers
Distinct chain representation
560
Distinct chain representation

Fig. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/041150

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/30(2006.01)i, 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 17/30; G06F 15/16; H04L 29/08; G06F 17/00; A63F 13/60; A63F 13/79

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: time-varying data, adjacency list, graph, relationship, distinct chain, 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.
X	US 2014-0040378 A1 (FACEBOOK, INC.) 06 February 2014 See paragraphs [0020]-[0028] and [0032]-[0035]; claims 21, 28, and 35; and figures 1-3.	1-15
A	US 2014-0122586 A1 (THE BOEING COMPANY) 01 May 2014 See paragraphs [0056]-[0069] and [0082]-[0089]; and figures 2 and 4-5.	1-15
A	US 2015-0011320 A1 (ZYNGA INC.) 08 January 2015 See paragraphs [0038]-[0039] and [0043]-[0046]; and figures 1-2 and 6A-6B.	1-15
A	WO 2013-131108 A1 (LINKEDIN CORPORATION) 06 September 2013 See paragraphs [0016]-[0020], [0026], and [0031]-[0036]; and figures 1-3.	1-15
A	US 8,954,441 B1 (LINKEDIN CORPORATION) 10 February 2015 See column 3, lines 4-62; column 5, lines 35-64; and figures 1-2 and 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
20 April 2016 (20.04.2016)	**20 April 2016 (20.04.2016)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 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/US2015/041150

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2014-0040378 A1	06/02/2014	US 2005-267940 A1	01/12/2005
		US 2011-099167 A1	28/04/2011
		US 2012-078957 A1	29/03/2012
		US 8572221 B2	29/10/2013
US 2014-0122586 A1	01/05/2014	US 8862662 B2	14/10/2014
US 2015-0011320 A1	08/01/2015	EP 2482946 A1	08/08/2012
		US 2011-0212770 A1	01/09/2011
		US 2011-0212784 A1	01/09/2011
		US 2011-0213716 A1	01/09/2011
		US 2012-0028713 A1	02/02/2012
		US 2012-0030123 A1	02/02/2012
		US 2012-0071236 A1	22/03/2012
		US 2012-0077596 A1	29/03/2012
		US 2012-0078394 A1	29/03/2012
		US 2012-0078395 A1	29/03/2012
		US 2013-0084927 A1	04/04/2013
		US 8241116 B2	14/08/2012
		US 8317610 B2	27/11/2012
		US 8333659 B2	18/12/2012
		US 8771062 B2	08/07/2014
		WO 2011-041566 A1	07/04/2011
WO 2013-131108 A1	06/09/2013	CN 103502975 A	08/01/2014
		DE 212013000002 U1	06/09/2013
		EP 2673718 A1	18/12/2013
		US 2013-0254303 A1	26/09/2013
US 8954441 B1	10/02/2015	US 2015-0186459 A1	02/07/2015
		WO 2015-102659 A1	09/07/2015