



US009569558B1

(12) **United States Patent**
Bigelow et al.

(10) **Patent No.:** **US 9,569,558 B1**
(45) **Date of Patent:** **Feb. 14, 2017**

(54) **METHOD FOR BACKFILLING GRAPH
STRUCTURE AND ARTICLES COMPRISING
THE SAME**

(71) Applicant: **International Business Machines
Corporation**, Armonk, NY (US)

(72) Inventors: **Alex R. Bigelow**, Salt Lake City, UT
(US); **Megan Monroe**, Somerville, MA
(US)

(73) Assignee: **INTERNATIONAL BUSINESS
MACHINES CORPORATION**,
Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 0 days.

(21) Appl. No.: **14/951,975**

(22) Filed: **Nov. 25, 2015**

(51) **Int. Cl.**
G06F 7/00 (2006.01)
G06F 17/30 (2006.01)

(52) **U.S. Cl.**
CPC ... **G06F 17/30958** (2013.01); **G06F 17/30377**
(2013.01)

(58) **Field of Classification Search**
CPC G06F 17/30958
USPC 707/798, 809
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,750,864 B1 * 6/2004 Anwar G06F 17/30489
345/440
7,039,631 B1 * 5/2006 Finger, II G06F 17/30675

8,577,911 B1 * 11/2013 Stepinski G06F 17/30
707/765
2010/0189316 A1 * 7/2010 Walch G06K 9/0008
382/125
2013/0151536 A1 * 6/2013 Akoglu G06F 17/30327
707/748
2014/0351564 A1 11/2014 Bekas et al.
2016/0092527 A1 * 3/2016 Kang G06F 17/30958
707/756

OTHER PUBLICATIONS

GraphTrail: Analyzing Large Multivariate, Heterogeneous Net-
works while Supporting Exploration History, Cody Dunne et al.,
CHI 2012, May 5-10, 2012, Austin, TX, USA, ACM.
Dynamic Data Structures for Document Collections and Graphs, J.
Ian Munro et al., PODS' 15, May 31-Jun. 4, 2015, Melbourne,
Victoria, Australia, ACM.

* cited by examiner

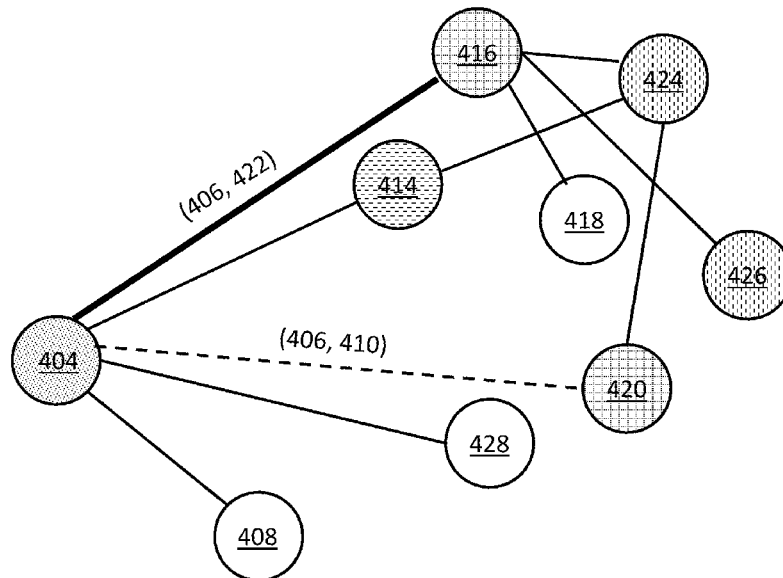
Primary Examiner — Cheyne D Ly

(74) *Attorney, Agent, or Firm* — Cantor Colburn LLP;
Kevin Jordan

(57) **ABSTRACT**

Disclosed herein is system for updating a graph-based data
structure comprising a processor that hosts an algorithm to
incrementally update underlying data and abstractions for a
graph-based data structure based on popular user selections;
a user interface that is in two-way operative communication
with the processor; where the user interface is operative to
present graph-based data structures to a user for updating;
and a memory database that is in communication with the
processor and is adapted to receive, store, and recall data
to/from the processor.

18 Claims, 9 Drawing Sheets



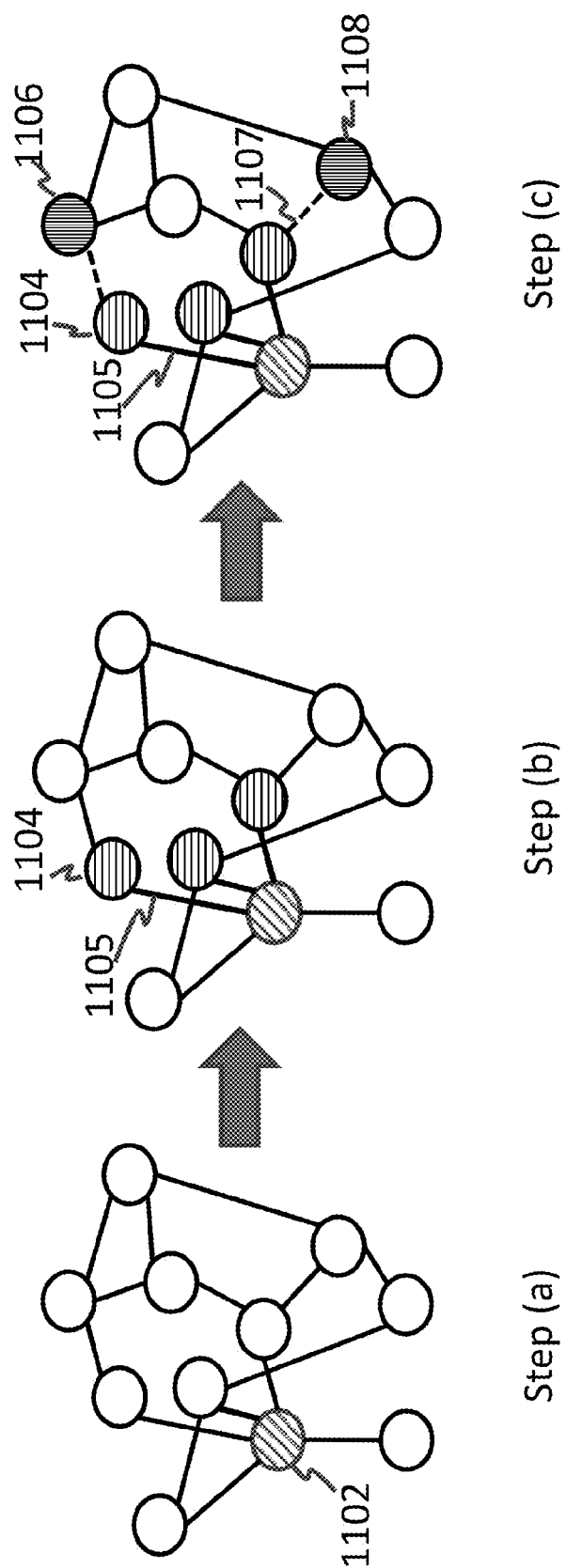


Figure 1 (Prior Art)

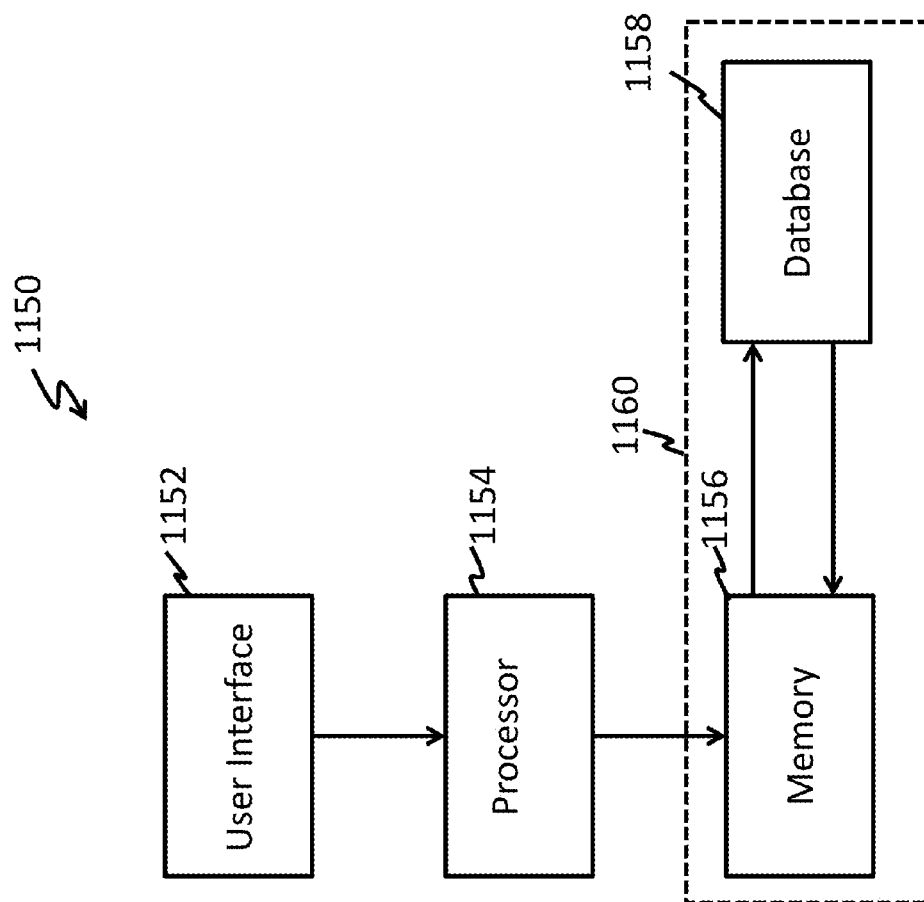


Figure 2

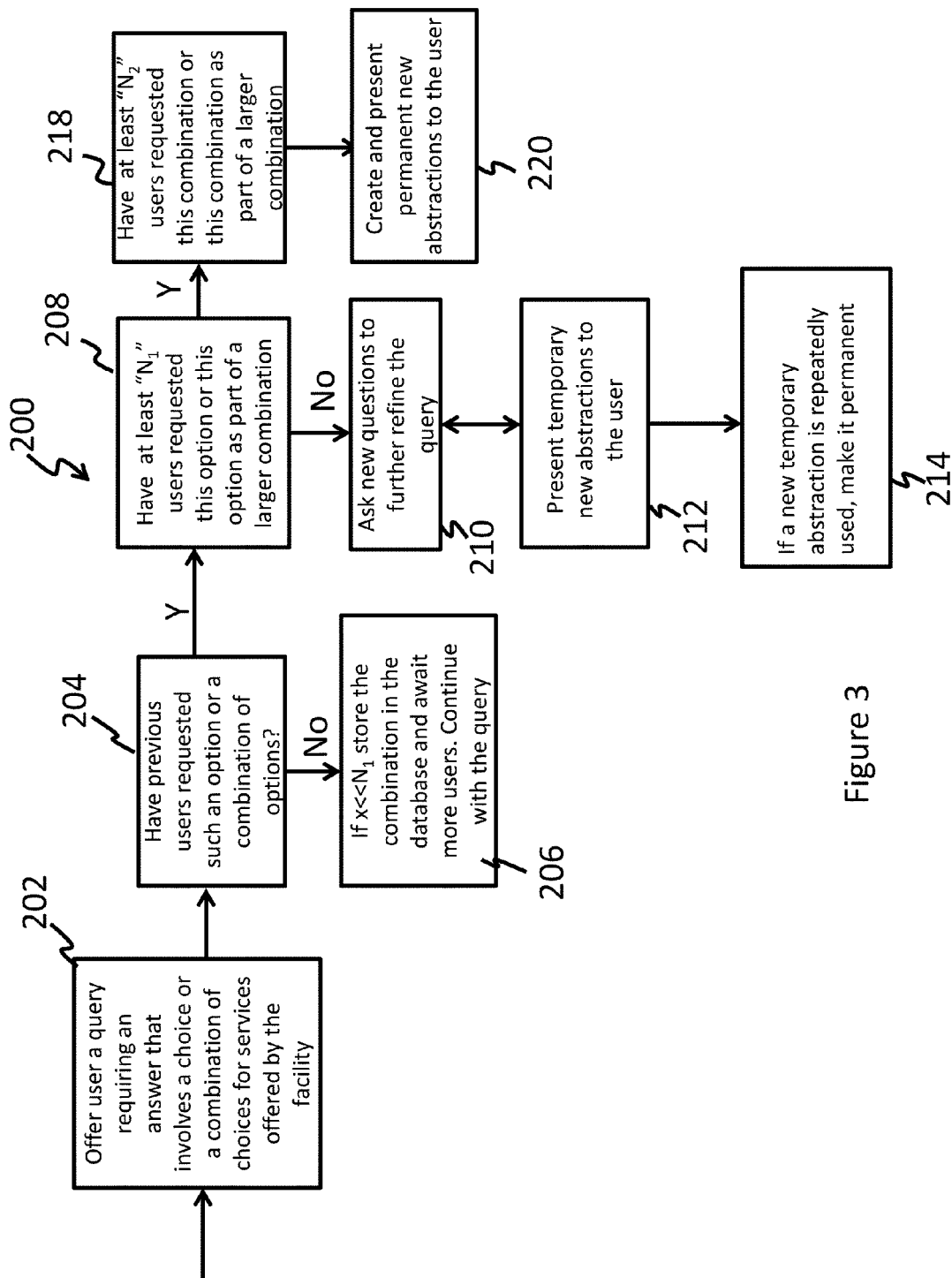


Figure 3

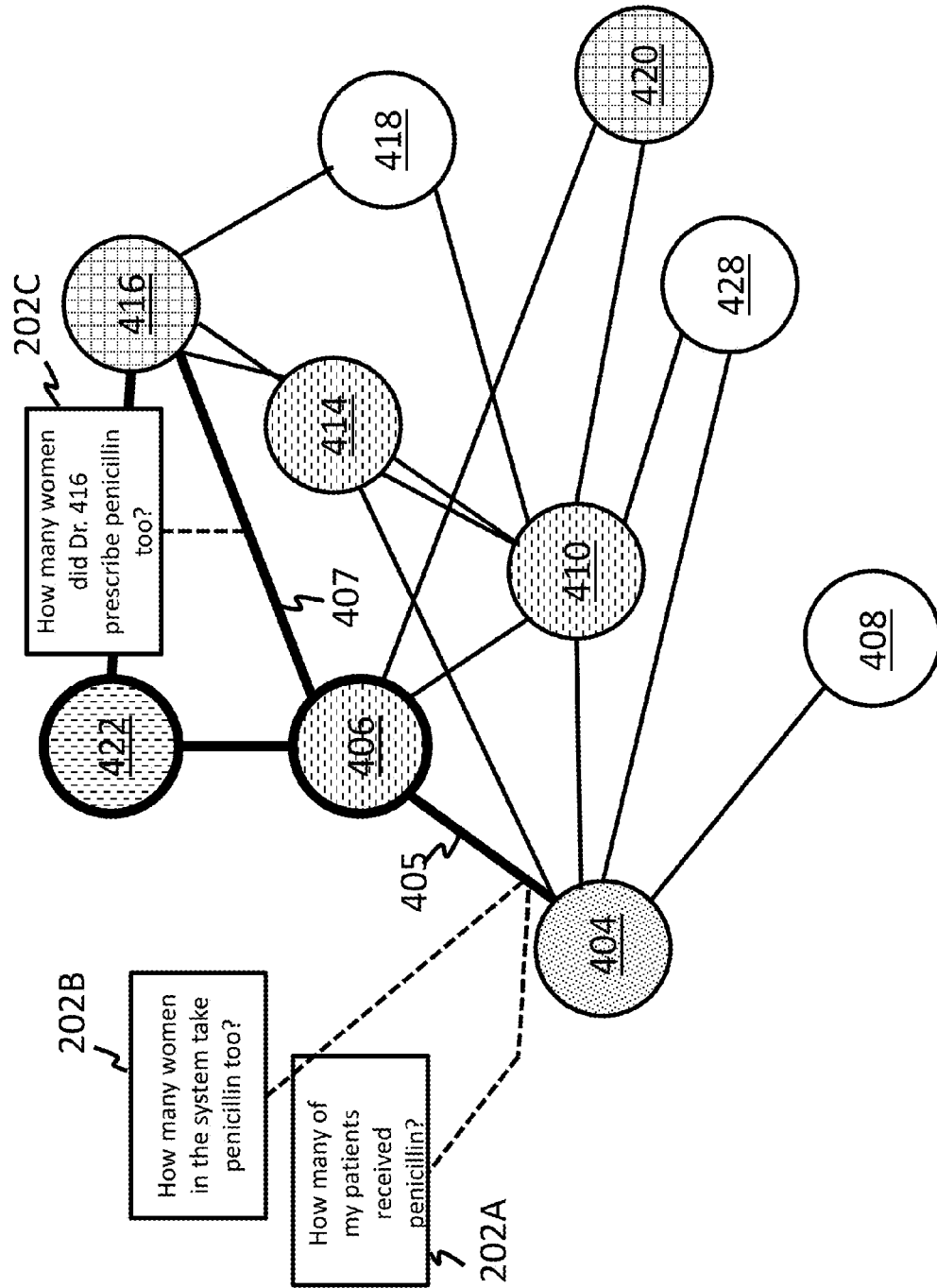


Figure 4

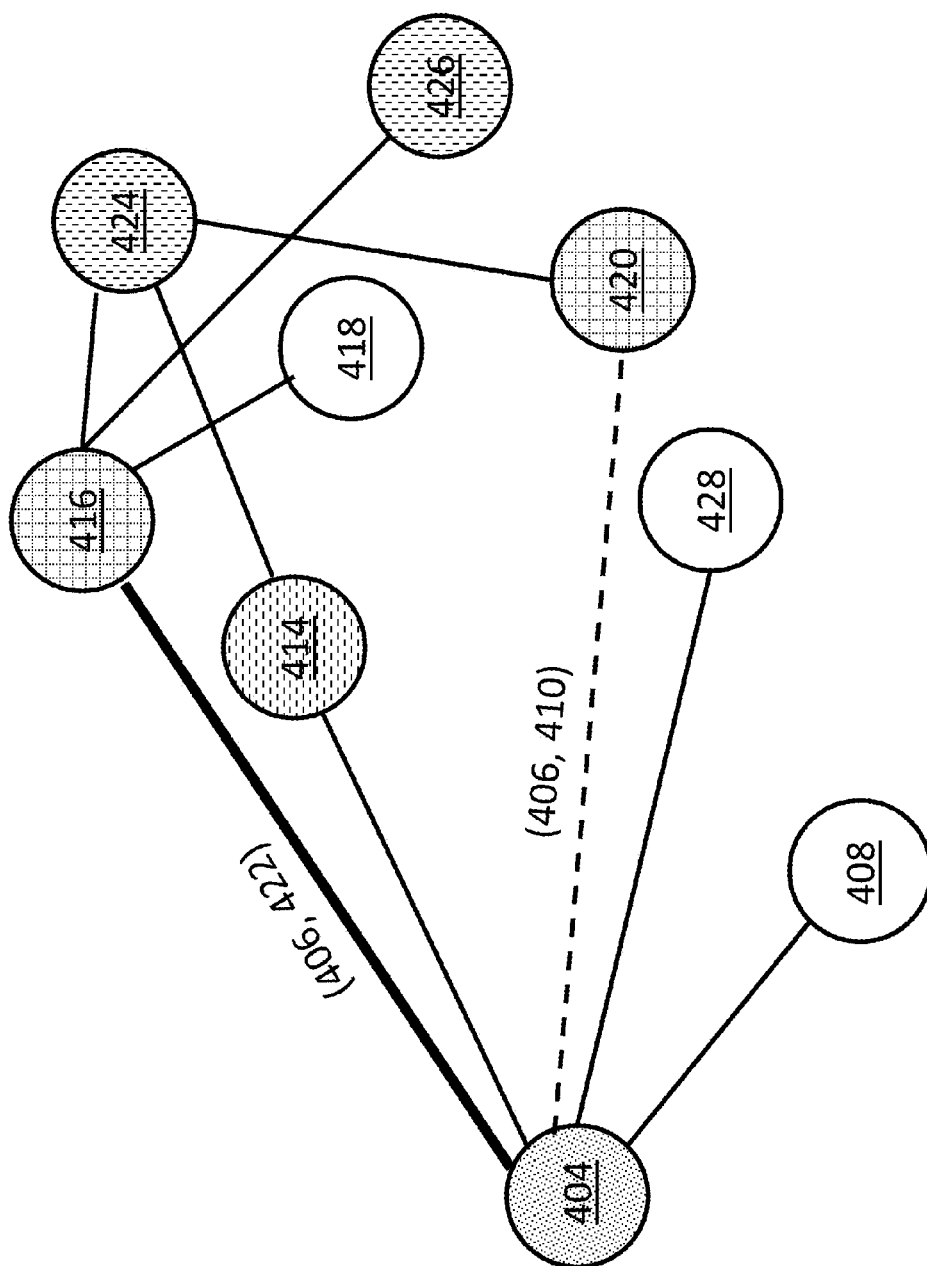


Figure 5

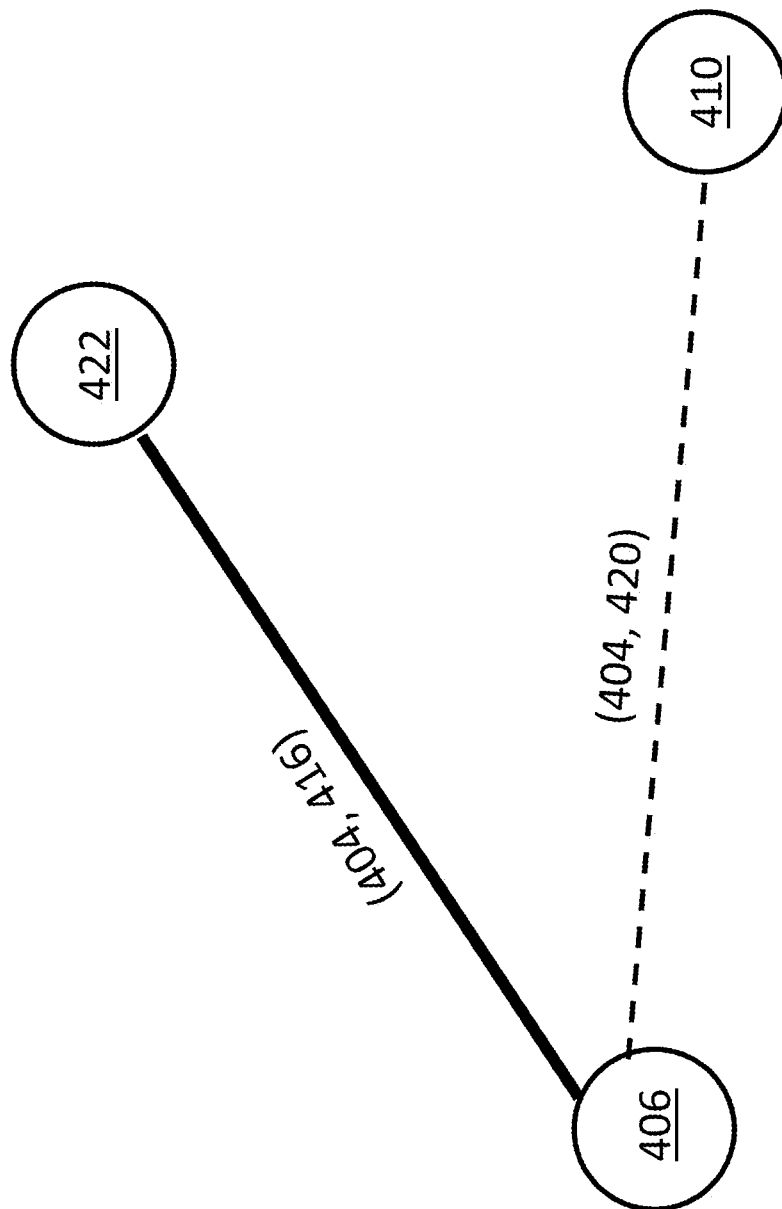


Figure 6

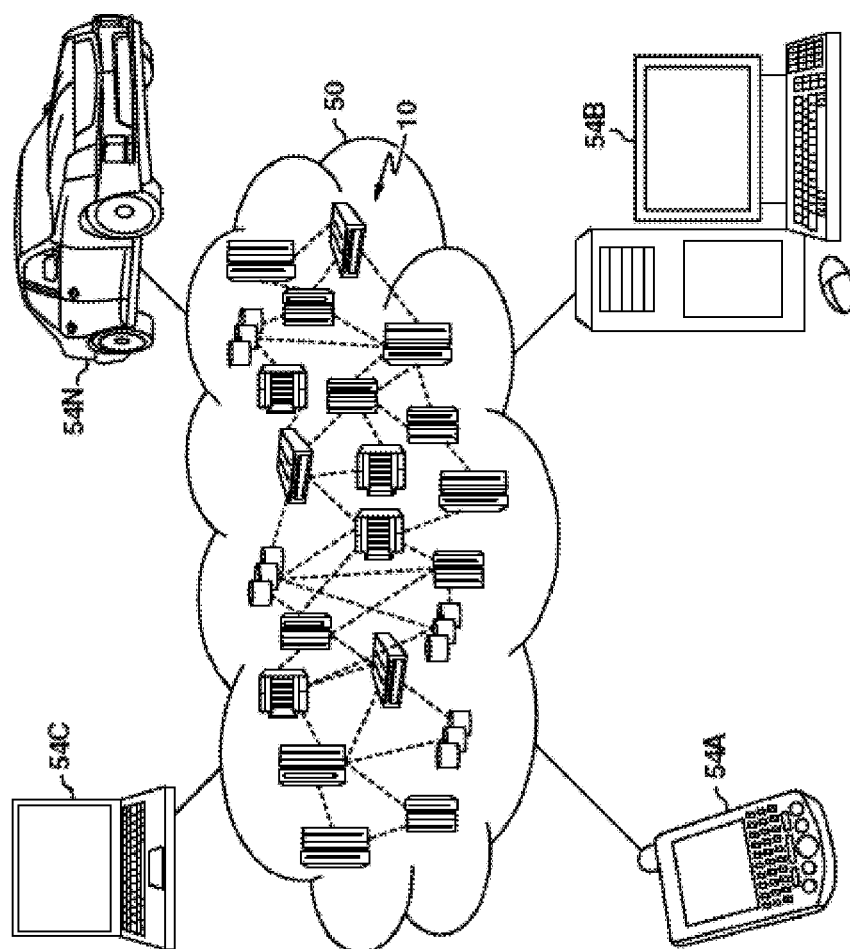


Figure 7

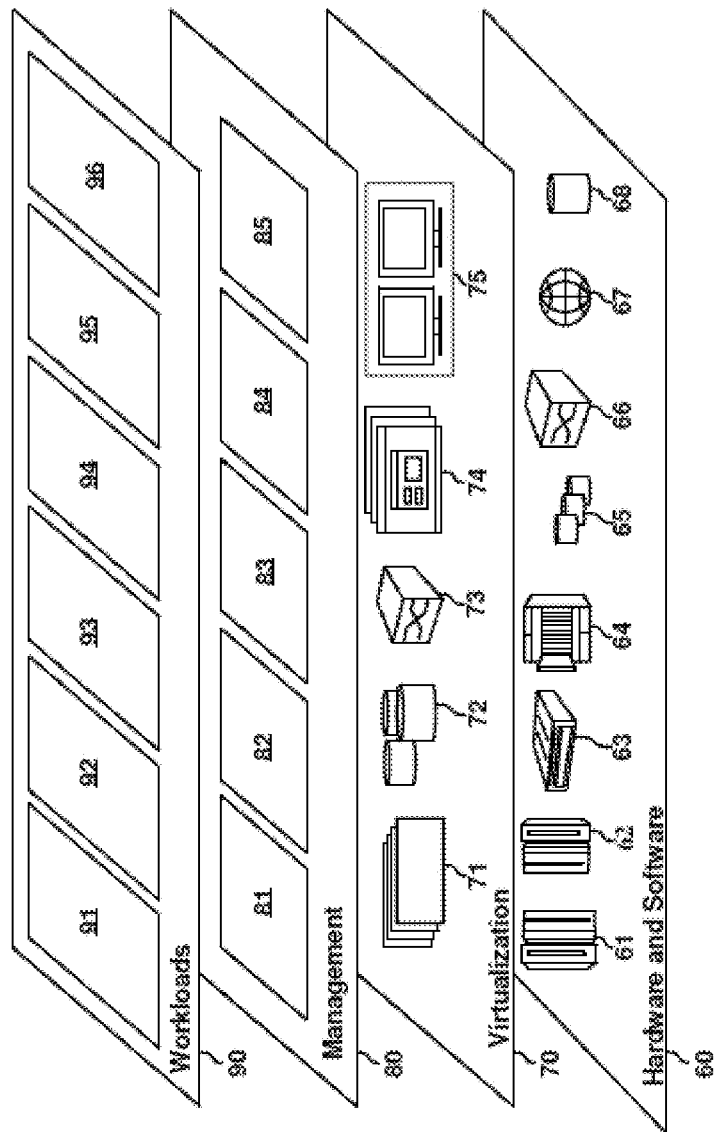


Figure 8

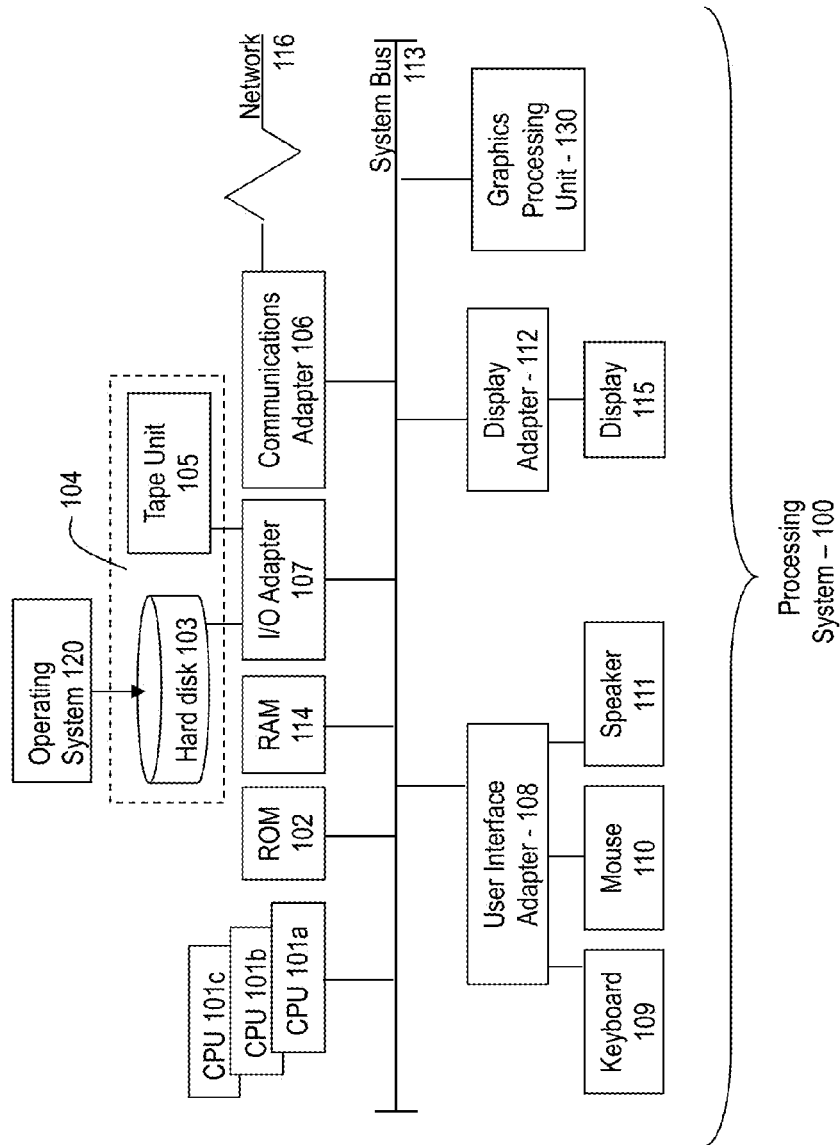


Figure 9

1

METHOD FOR BACKFILLING GRAPH STRUCTURE AND ARTICLES COMPRISING THE SAME

BACKGROUND

This disclosure relates to a method for backfilling graph structure and to articles comprising the same. In particular this disclosure relates to a method for generating new graph-based data structures and adding them to existing graph-based data structures based on queries posed by users of the system over a period of time.

Graph-based data systems provide information in form of nodes and edges in a wide variety of data systems such as, for example, those used in hospitals, police and detective databases, university systems, employment databases, city service databases, and the like. Graph-based data was once thought of as a fallback option for data that could not be manipulated into a relational data system. However, graph-based data structures and graph-based data systems are now emerging as the preferred storage method, not only for overtly networked systems, such as social networks and citation networks, but also for biological systems, traffic patterns and, well, all of human knowledge.

The FIG. 1 shows how moving along the nodes and edges of a graph-based structure can be used to support a variety of different queries that users may need to execute. The term “graph-based” data structure as used herein refers to a data structure comprised of nodes and edges. Nodes represent entities such as people, businesses, accounts, or any other item one might want to keep track of. Edges are the lines that connect nodes to nodes or alternatively, nodes to properties and they represent the relationship between the two. Meaningful patterns emerge when examining the connections and interconnections of nodes, properties, and edges.

For example, in the data system for a large hospital, a doctor **1102** might want to find out which other doctors his patients are seeing. By first finding himself **1102** in the data system (see step (a)), he can then pivot out to all of the patients **1104** associated with him (see step (b)), and then pivot back to all of the doctors **1106** and **1108** associated with those patients (see step (c)). A pivot is the process of selecting an initial set of seed nodes (in this case “the doctor **1102**”) in the graph, and then swinging out to the neighboring nodes (in this case “all of the patients **1104**”) that are connected to it. This produces subgraph data consisting of both the seed node (**1102**) and neighbor nodes (**1104**). The term “pivot” comes from the fact that this operation can be chained together, with the neighbor nodes (**1104**) from the previous step serving as the seed nodes in the present step to determine another set of neighbor nodes (**1106** and **1108**).

In the FIG. 1, the lines (shown in bold) connecting the doctor **1102** to his patients **1104** are called edges. These bold lines represent a first set of edges **1105**. The lines connecting the patients **1104** to their other doctors **1106** and **1108** are shown in dotted lines and these represent a second set of edges **1107**. While the exemplary graph shown in the FIG. 1 is helpful to the doctor to determine which other doctors service some of his patients, not all users of the system will find the graph-based data system easy to access and even fewer will find the means (e.g., querying the system using keywords approved by the system) to access the structure and to obtain all information available to them from the structure.

It is therefore desirable to have a graph-based data structure where abstractions are developed and continuously improved based on the type and population of queries that

2

the system is subjected to over time by users and not just by the people that create the abstractions.

SUMMARY

Disclosed herein is system for updating a graph-based data structure comprising a processor that hosts an algorithm to incrementally update underlying data and abstractions for a graph-based data structure based on popular user selections; a user interface that is in two-way operative communication with the processor; where the user interface is operative to present graph-based data structures to a user for updating; and a memory database that is in communication with the processor and is adapted to receive, store, and recall data to/from the processor.

Disclosed herein too is a method comprising traversing a graph-based data structure comprising edges and nodes along at least one edge or node on a user interface in a system comprising a processor that hosts an algorithm to incrementally update underlying data and abstractions for a graph-based data structure based on popular user selections; the user interface; where the user interface is in two-way operative communication with the processor; where the user interface is operative to present graph-based data structures to a user for updating; and a memory database that is in communication with the processor and is adapted to receive, store, and recall data to/from the processor; and incrementally updating the graph-based data structure.

Disclosed herein too is a computer program product comprising a non-transitory storage medium readable by a processing circuit and storing instructions for execution by the processing circuit for performing a method comprising traversing a graph-based data structure comprising edges and nodes along at least one edge or node on a user interface in a system comprising a processor that hosts an algorithm to incrementally update underlying data and abstractions for a graph-based data structure based on popular user selections; the user interface; where the user interface is in two-way operative communication with the processor; where the user interface is operative to present graph-based data structures to a user for updating; and a memory database that is in communication with the processor and is adapted to receive, store, and recall data to the processor; and incrementally updating the graph-based data structures.

BRIEF DESCRIPTION OF THE FIGURES

FIG. 1 shows how a graph-based data structure with nodes and edges may be used to determine a variety of different options available to a user;

FIG. 2 is an exemplary schematic depiction of the components of the graph-based data structure;

FIG. 3 is a flow diagram that details one exemplary sequence of events by which the graph-based data structure uses movements and queries of the use to generate new abstractions;

FIG. 4 depicts one embodiment of a presentation of a graph-based data structure that may be presented to the user as a result of new abstractions generated by the system;

FIG. 5 depicts an embodiment of the FIG. 3 where the criteria for edges is changed;

FIG. 6 depicts an embodiment of the FIG. 3, where nodes and edges are flipped;

FIG. 7 is illustrative of a cloud computing environment;

FIG. 8 depicts a set of functional abstraction layers provided in the cloud computing environment; and

FIG. 9 depicts an embodiment of a processing system for implementing the teachings herein.

DETAILED DESCRIPTION

Disclosed herein is a continuously self-improving system for graph-based data structures (also interchangeably referred to herein as the “system”) that develops new abstractions for the graph-based data structure based on user interactions with the system. Disclosed herein too are methods by which the graph-based data structure continuously updates its abstractions based on user interactions with the system. The system disclosed herein queries users as they traverse the data structure along its nodes and edges and uses the answers to extract data for the user, to generate new abstractions and to update old abstractions. The movements and the queries of the user are recorded and aggregated and then used to algorithmically determine improvements to existing data abstractions or to generate new abstractions and new graph-based data structures. These improvements can take the form of adding new edges to the graph-based data structure, reformulating node attributes as nodes themselves, converting nodes to edges and vice versa, to list a few improvements. The end result is a graph-based data structure that can evolve its underlying abstractions to better suit the tasks and queries that data already present in the system can answer.

The graph-based data structure detailed herein provides the user with a means of incrementally, but freely moving through the graph data. By using the pivot one can select an initial set of seed nodes in the graph data, and then swing out to the neighboring nodes that are connected to the set of seed nodes to traverse the graph data. This produces a subgraph consisting of both the seed and neighbor nodes. These pivots swing out to all of the neighboring nodes, or only to nodes of a certain category, where the user can perform a wide range of filtering on the subgraph before the next pivot is performed. For example, in a medical database, a user might filter a list of all the doctors down to just the female doctors and then pivot out to see their patients. Using the pivot operation, users can move across existing edges in the graph to formulate queries, extract data, and understand how elements are connected. The graph-based data structure observes these sequences of pivots and filtering’s across the entire user base, and identifies common patterns of usage. When a pattern is executed frequently, the system automatically adapts the underlying data structure to better support that operation.

With reference now to the FIG. 2, a system 1150 for hosting and deploying the graph-based data structure comprises a user interface 1152, a processor 1154, and a memory database 1160 that comprises a memory 1156 and a database 1158.

The user interface 1152 is in two-way operative communication with the processor 1154 that hosts an algorithm to incrementally update the underlying data and abstractions in the graph-based data structure. The user interface 1152 can comprise a monitor, a speaker, or any other devices that permit speech to text and/or text to speech communication with the graph-based data structure. The user interface 1152 may provide queries to the user in text and/or speech format and may display the generated text resulting from the speech-to-text conversion to facilitate interaction with the system 1150. The user interface 1152 also permits the user to correct the text and to provide other contextual details to the system either via an aural feedback or a textual feedback loop. The user interface may be in communication with the

system via one or more networks such as, for example, a wireless cellular network, a wireless local area network (WLAN) or Wi-Fi network, a Third Generation (3G) mobile telecommunications network, a private network such as an intranet, a public network such as the Internet, or some combination thereof.

The memory 1156 in conjunction with the database 1158 stores abstractions and the resulting graph and subgraph-based data structures. It also stores queries that are asked of the user and queries that the user may ask as he/she traverses a graph. The memory 1156 is in communication with the processor 1154 and adapted to receive, store, and recall data to the processor 1154. The processor hosts an algorithm that is operative to permit updates and modifications to graph-based data structures and to abstractions, upon which the graph-based data structures are created, based broadly on changes initiated by users. The memory 1156 is in operative communication with the database 1158 and adds corrections or changes to the database or alternatively gets new data structures and abstractions from the database 1158. The memory 1156 stores unused queries, users patterns of access and works cooperatively with the processor 1154 to develop new queries, abstractions and graph-based data structures. In an embodiment, the memory 1156 and the correction database 1158 can be part of a single system, i.e., they can be a part of a single unitary device 1160 as seen in the FIG. 1(B).

The memory 1156 may be provided with an array of records and parameters that can be accessed and utilized in order to produce the graph and subgraph-based data structures. An exemplary system for accessing graph-based data structures and transforming nodes and edges of a particular graph-based data structure based on new methods of using the system is provided in the FIG. 3.

FIGS. 3 and 4 detail one exemplary sequence of events by which the graph-based data structure 200 uses movements and queries of the use to generate new abstractions. The FIG. 3 is a flow diagram (an algorithm) that depicts how the system 200 interacts with the user. The FIG. 4 depicts a graph-based data structure that shows how moving along the edges or to the nodes of the data structure may be used to make further inquiries of the user to develop new abstractions and new graph-based data structures that make use of the system easier. It is to be noted that the FIGS. 4, 5 and 6 in this disclosure represent what is happening in the underlying data structure and not necessarily what the users are seeing on the user interface. There are many possible graphical interfaces that would allow users to specify queries against the underlying data structure, but the main idea behind this invention is that—regardless of what the user interface is—queries are being used to modify the data structure, not necessarily just the interface.

With reference now to the FIG. 3, when a user activates the graph-based data structure 200, he or she may enter his or her name into the database via a user interface if the user has previously interacted with the system 200. It is to be noted that the system is not restricted just to prior users but can be activated by new users as well. New users may be requested to provide some information into the system 200 to facilitate the activation of the queries (not shown).

Upon activating the system 200, the system may optionally present queries 202 to the user. The FIG. 3 is broadly based on the principle that the graph and the queries presented to the user (for the purposes of developing further abstractions in the system 200) are dependent upon the amount of use of the queries made by preceding users and by the manner in which users traverse the graph-based data structure looking for additional information. In short, if a

5

query is particularly popular with users, the data obtained from the answers to this query are more likely to be used to update the abstractions and the graph-based data structure than another query that is not as popular with the users. In another embodiment, if a plurality of users navigate a particular graph-based data structure to find desired information using a particular combination of nodes, edges and pivots, then the abstractions for the graph data system are updated to reflect this particular combination. When another user seeks to find the desired information, the updated graph-based data structure presented to the new user is more intuitive than the old graph-based data structure and this makes finding the information more easy.

For example, if a particular query results in a particular option being chosen N1 times by a plurality of users, the system may develop additional queries around this particular option. It may also start to develop new graph-based data structures around this particular option and present them to a limited number of users for beta-testing. On the other hand, if a particular query results in a particular option being selected N2 times, where N2 is greater than N1, then the system presents the next user with an already developed graph-based data structure that is the result of new abstractions generated from queries and options selected by the previous N2 users. Alternatively, if very few users (x in number) chose a particular option, then the option is saved to the memory database and is offered to addition users for a period of time. In an embodiment, x is far less than N1 and as x approaches N1, the system may start to develop additional queries around the option and may begin graph-based data structure development for beta-testing with further users of the system **200**.

The queries **202** (see FIGS. 3 and 4) may cause the user to make a choice of the data that he/she would like to access. Some users who are already familiar with the system **200** may make their choices known to the system as soon as they sign on and may therefore not have to be presented with the initial queries. While only a single query **202** is shown in the FIG. 3, the system may make a plurality of queries such as **202A**, **202B**, **202C**, and so on (see FIG. 4). The user can stop the queries whenever he/she desires. The queries may usually require an answer from the user that involves selecting an option or a combination of options available from the system **200**. Based on the option selected by the user, the system **200** may present a new graph-based data structure to the user.

When, for example, doctor **404** in the FIG. 4 activates the system it displays a seed node **404** that pertains to him. He now queries the system as to how many female patients he has. The system displays a graph-based data structure having 5 female patients to him as neighboring nodes—**406**, **408**, **410**, **414** and **428**. Connecting lines (edges) represents connections between the nodes (**406**, **408**, **410**, **414** and **428**) representing his patients and the node **404** that represents him. The doctor **404** now queries the system **200** about other doctors that his female patients are seeing. The graph-based data structure is updated with new nodes **416** and **420** that represent the doctors that his female patients are seeing. By pivoting from one set of nodes (the female patients) to another set of nodes (their doctors), the doctor **404** is able to traverse the system.

Another patient **418** is also seeing doctor **416**. Doctor **404** is now desirous of determining the treatment his patients are getting from the other doctors **416** and **420**. As he moves along edge **405** from his node **404** to node **406** that represents one his patients, the system **200** begins querying him. Query **202A** (see FIGS. 3 and 4) asks him if he is interesting

6

in knowing how many of his women patients are taking antibiotics (penicillin) while query **202B** asks him whether he wants to know how many women in the system **200** take antibiotics (penicillin) too. If the doctor chooses “yes” to question **202A**, then nodes **406**, **410** and **414** are immediately colored indicating that three of his female patients **406**, **410** and **414** are taking antibiotics. If he answers “yes” to query **202B**, then the system adds additional women to the graph in the form of a node—**422**. All the nodes representing women who have received antibiotics from the doctors in the system **200** have edges **405** and **407** (shown in bold) connecting them to node **416** and node **404** because these doctors have both prescribed antibiotics. Doctor **404** can now traverse the edge between node **406** (his patient **406**) and node **416** (doctor **416**) along edge **407**. During this traversal, the system may query him once again (see query **202C** in the FIG. 3) as to whether he wants to know how many of these women doctor **416** has prescribed antibiotics too. If he answers yes, the system **200** highlights nodes **406** and **422** (see bolded circles) as being patients that doctor **416** prescribed antibiotics to. It is to be noted that nodes and edges described herein pertain to the underlying data structure and not to the graphical data seen on the screen of the user interface.

Alternatively, as doctor **404** traverses the edge **405** between nodes **404** and **406**, he may want to query the system **200** about some other issues pertaining to nodes **406** and **416**. The doctor **404** may want to know which insurance companies pay for antibiotics and how much they pay. These queries (not shown) may be stored in the system and aggregated for further use in developing either additional queries, abstractions or graph-based data structures. Doctor **404** may also perform filtering steps to make further determinations of other features of his patients. For example, he may want to know which of his female patients who have taken antibiotics are over 50 years of age.

By using a combination of pivots and filtering steps, making inquiries and answering queries the doctor **404** enables the graph-based data structure **200** to make new abstractions which result in new data structures that can help future users easily navigate the data structure and derive pertinent information in a timely manner.

With reference now to the FIGS. 2 and 3, if a user selects an option, the system **200** examines the memory database **158** to determine if previous users have requested such an option, as shown at block **204**. If the answer is yes the system **200** moves to step **208** where it determines if at least N1 users have requested this exact option or have requested this option as part of a bigger combination of options. Step **208** and those steps **210**, **212** and **214** that depend from it will be detailed later.

If the answer at step **204** is “no”, then the system **200** tries to determine if the number (x) of a particular option chosen by the users is far below a certain threshold number (N1). If so, then the option is stored in a memory database **206** of the system **200**. In other words, if $x \ll N1$, the option is stored in the memory database and reused as a query without developing any new abstractions based on it. As seen previously (with respect to the FIG. 3) as the user moves along the edge of the graph-based data structure more queries are presented to the user. Each of these queries is stored in the memory of the database and reused if $x \ll N1$. If the number (x) of choices for a particular query exceeds N1 but is less than N2, where N2 is greater than N1, then the system moves to step **208**.

If the answer to the query in step **208** is “no”, then the system **200** attempts to ask new queries (see step **210**) based

around the old query to further refine the query. The system 200 may solicit help from users to refine the query. The system performs step 210 because “x” is tending to “N1”. In other words, the answers to queries begin to determine that there is appreciable interest in this particular query or line of reasoning from the various users of the system. New queries based on temporary new abstractions may also be presented to users in step 210. Step 210 and 212 can interact with each other. In other words, if the system 200 determines that the temporary abstractions of step 212 are gaining traction with users then it can present new queries in step 210 to test which queries and graph-based data structures are being populated by the users. In step 214, if the system notes that a particular graph-based data structure is being repeatedly used it can update the abstractions and make this new graph-based data structure permanent—in other words, the new graph-based data structure can be presented to users who have an interest in this particular line or reasoning.

In step 208 if the answer to the query “have at least N1 users requested this exact option or have they requested this option as part of a combination with other options?” is yes then the system moves on to step 218 where it checks to see if more than N2 users are availing of this option. N2 is always greater than N1 and represents another threshold number. If more than N2 users are using a particular option then the abstraction is made permanent (see step 220) and a new graph-based data structure based on this abstraction is always presented to a user desirous of obtaining information from the system. In this example, it should be noted that x, N1 and N2 are integers.

In summary, if a large set of users of the graph-based data structure explore a particular data structure in a hitherto unused manner, the system develops new abstractions and consequently new data structures to enable future users to better navigate the system and extract information contained therein.

While the criteria used in the FIG. 3 to develop new graph-based data structures is based on the number of users who choose a particular option versus other available options, other criteria may also be used to update graph-based data structures. Exemplary criteria can be the rate of usage of a particular option between particular time periods (e.g., the hours of 11:00 AM to 3:00 PM), the activity during a particular part of the year (e.g., the summer versus the winter), degree of difficulty of course work (e.g., in a university or school). Other criteria not listed here can also be used to create thresholds for updating the data structures.

In an embodiment, it may also be possible to identify expert users (perhaps users who understand both technology and the particular domain that the data pertains to) and place a higher value on their patterns of use. Alternatively, if there is some measure of a user’s success within the system (whether they found what they were looking for in a given time period), the patterns of more successful users can be assigned a higher value. These values are the aggregated and used in making a determination of whether to update the graph-based data structure.

In some situations it may be desirable to rearrange the relationship between nodes and edges so as to make the graph-based data structure more intuitive and to make searching for relationships in a particular course of action more easy. One manner of accomplishing this is by turning some of the nodes into edges. FIG. 5 reflects one such iteration. The FIG. 5 uses the graph-based data structure from the FIG. 4. In the FIG. 4, patients 406 and 410 are

connected to doctors 404, 416 and 420. In addition, patients 406 and 422 have been prescribed antibiotics by doctors 404 and 416.

By flipping the nodes into edges of the graph-based data structure and removing the extraneous edges that do not play any role in this particular graph-based data structure, the data structure can be simplified to that shown in the FIG. 5. In the FIG. 4, the bold edge labelled (406, 422) connects nodes 404 and 416 while the dotted edge (406, 410) connects nodes 404 and 420 because they are serviced by doctors 404 and 420. The graph-based data structure shown in the FIG. 4 is substantially simpler than that seen in the FIG. 3. The bolded edge labelled (406, 422) indicates that patients 406 and 422 have received antibiotics from doctors 404 and 416.

If a large number of queries are being directed to the system 200 by users wanting to know particulars about only the use of antibiotics by doctors 404 and 416 and other doctors that see the patients who take antibiotics, the system may be further simplified to that shown in the FIG. 6. In the FIG. 6, the nodes that represented the patients 406, 410 and 422 have now become primary nodes while the doctors 404, 416 and 420 are now represented by the edges of the graph-based data structure. Edge (404, 416) is bolded because it represents doctors who prescribe antibiotics, while edge (404, 420) is dotted because it represents another doctor 420 that has worked on one of doctor 404’s patient (410). The FIG. 6 can be seen to be significantly simpler and more intuitive when the attributes of nodes and edges from the FIGS. 4 and 5 are changed.

The result of these alterations to the underlying data structure is that the graph-based data structure can adapt to better support current and new questions. The system learns which connections hold the most valuable, real-world knowledge and exposes those connections as directly as possible. These updates are performed automatically, either as relevant patterns are detected, or as the processing and storage resources become available to support the added complexity. Overall, the system allows the underlying data abstraction to be improved in situ, without constant collaboration between the technical experts and the domain experts. Using this method of backfilling structure, the graph-based data structure automatically adapts to be able to efficiently deliver what users need from it. The system may be used in variety of operations such as hospitals, police departments, homeland security departments, school and university systems, agricultural planning, and the like. The system may be used for cases including matchmaking, network management, software analytics, scientific research, routing, organizational and project management, recommendations, social networks, and more.

The graph-based data structure 200 disclosed herein is exemplified by the following non-limiting examples.

Example 1

A city, for example, may be viewed as an attribute of say, a university node (i.e., the city in which that university is located). However, in some cases it might make more sense for each city to be its own node in the graph, and for the location of a university to be represented by an edge to that city node. Flipping the notion of nodes and edges entirely can produce a more intuitive graph. In a similar manner revisiting the criteria (the abstractions) for what constitutes a node and an edge can produce more intuitive graph-based data structures. Using the manner in which users use a

graph-based data structure to create new criteria is another way of producing more intuitive graphs.

Example 2

This example deals with doctors. Suppose that doctors must frequently determine which treatments can be prescribed based on a patient's insurance provider. However, in the initial graph abstraction, insurance providers and treatments are only connected through patients. The system observes users performing frequent pivots from treatments, to patients, to insurance providers, and back. When this pattern reaches a certain threshold of usage, the system automatically adds a set of edges that directly connect the insurance providers with the prescribed treatments. This allows users to move directly between these elements, and make correlations without having to pivot through the patient nodes.

Example 3

Suppose that administrators at a university are frequently trying to pair students with professors from their home country. In the data system, the home countries of both students and professors are stored as an attribute of those nodes. The system observes users frequently using this attribute to correlate these two types of nodes. In response the system pushes the country attribute out into the graph-based data structure as actual nodes, allowing users to make direct pivots between students and professors from the same country.

In accordance with exemplary embodiments of the disclosure, methods, systems and computer program products for cloud storage of mobile applications are provided. In exemplary embodiments, a mobile device, such as a smartphone or tablet, is configured to selectively store one or more applications in a cloud storage system to increase an available local storage on the mobile device. The mobile device is further configured to monitor the usage and context of the mobile device and to retrieve applications from the cloud storage system based on an anticipated usage of the applications by the mobile device. In exemplary embodiments, the storage and retrieval of applications in the cloud storage system is configured such that the user is not aware that the application has been stored remotely. In addition, any user data or customizations of the application installed on the mobile device are configured to remain on the mobile device, such that no change in the function or appearance of the application is notice when the application is stored in and retrieved from the cloud storage system.

It is understood in advance that although this disclosure includes a detailed description on cloud computing, implementation of the teachings recited herein are not limited to a cloud computing environment. Rather, embodiments of the present invention are capable of being implemented in conjunction with any other type of computing environment now known or later developed.

Cloud computing is a model of service delivery for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, network bandwidth, servers, processing, memory, storage, applications, virtual machines, and services) that can be rapidly provisioned and released with minimal management effort or interaction with a provider of the service. This cloud model may include at least five characteristics, at least three service models, and at least four deployment models.

Characteristics are as follows:

On-demand self-service: a cloud consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with the service's provider.

Broad network access: capabilities are available over a network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs).

Resource pooling: the provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to demand. There is a sense of location independence in that the consumer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter).

Rapid elasticity: capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

Measured service: cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported providing transparency for both the provider and consumer of the utilized service.

Service Models are as follows:

Software as a Service (SaaS): the capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based e-mail). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

Platform as a Service (PaaS): the capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including networks, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

Infrastructure as a Service (IaaS): the capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

Deployment Models are as follows:

Private cloud: the cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on-premises or off-premises.

Community cloud: the cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements,

11

policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on-premises or off-premises.

Public cloud: the cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

Hybrid cloud: the cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

A cloud computing environment is service oriented with a focus on statelessness, low coupling, modularity, and semantic interoperability. At the heart of cloud computing is an infrastructure comprising a network of interconnected nodes.

Referring now to FIG. 7, illustrative cloud computing environment 50 is depicted. As shown, cloud computing environment 50 comprises one or more cloud computing nodes 10 with which local computing devices used by cloud consumers, such as, for example, personal digital assistant (PDA) or cellular telephone 54A, desktop computer 54B, laptop computer 54C, and/or automobile computer system 54N may communicate. Nodes 10 may communicate with one another. They may be grouped (not shown) physically or virtually, in one or more networks, such as Private, Community, Public, or Hybrid clouds as described hereinabove, or a combination thereof. This allows cloud computing environment 50 to offer infrastructure, platforms and/or software as services for which a cloud consumer does not need to maintain resources on a local computing device. It is understood that the types of computing devices 54A-N shown in FIG. 7 are intended to be illustrative only and that computing nodes 10 and cloud computing environment 50 can communicate with any type of computerized device over any type of network and/or network addressable connection (e.g., using a web browser).

Referring now to FIG. 8, a set of functional abstraction layers provided by cloud computing environment 50 (FIG. 7) is shown. It should be understood in advance that the components, layers, and functions shown in FIG. 8 are intended to be illustrative only and embodiments of the invention are not limited thereto. As depicted, the following layers and corresponding functions are provided:

Hardware and software layer 60 includes hardware and software components. Examples of hardware components include: mainframes 61; RISC (Reduced Instruction Set Computer) architecture based servers 62; servers 63; blade servers 64; storage devices 65; and networks and networking components 66. In some embodiments, software components include network application server software 67 and database software 68.

Virtualization layer 70 provides an abstraction layer from which the following examples of virtual entities may be provided: virtual servers 71; virtual storage 72; virtual networks 73, including virtual private networks; virtual applications and operating systems 74; and virtual clients 75.

In one example, management layer 80 may provide the functions described below. Resource provisioning 81 provides dynamic procurement of computing resources and other resources that are utilized to perform tasks within the cloud computing environment. Metering and Pricing 82 provide cost tracking as resources are utilized within the cloud computing environment, and billing or invoicing for consumption of these resources. In one example, these

12

resources may comprise application software licenses. Security provides identity verification for cloud consumers and tasks, as well as protection for data and other resources. User portal 83 provides access to the cloud computing environment for consumers and system administrators. Service level management 84 provides cloud computing resource allocation and management such that required service levels are met. Service Level Agreement (SLA) planning and fulfillment 85 provides pre-arrangement for, and procurement of, cloud computing resources for which a future requirement is anticipated in accordance with an SLA.

Workloads layer 90 provides examples of functionality for which the cloud computing environment may be utilized. Examples of workloads and functions which may be provided from this layer include: mapping and navigation 91; software development and lifecycle management 92; virtual classroom education delivery 93; data analytics processing 94; transaction processing 95; and storage of applications for mobile devices 96.

Referring to FIG. 9, there is shown an embodiment of a processing system 100 for implementing the teachings herein. In this embodiment, the system 100 has one or more central processing units (processors) 101a, 101b, 101c, etc. (collectively or generically referred to as processor(s) 101). In one embodiment, each processor 101 may include a reduced instruction set computer (RISC) microprocessor. Processors 101 are coupled to system memory 114 and various other components via a system bus 113. Read only memory (ROM) 102 is coupled to the system bus 113 and may include a basic input/output system (BIOS), which controls certain basic functions of system 100.

FIG. 9 further depicts an input/output (I/O) adapter 107 and a network adapter 106 coupled to the system bus 113. I/O adapter 107 may be a small computer system interface (SCSI) adapter that communicates with a hard disk 103 and/or tape storage drive 105 or any other similar component. I/O adapter 107, hard disk 103, and tape storage device 105 are collectively referred to herein as mass storage 104. Operating system 120 for execution on the processing system 100 may be stored in mass storage 104. A network adapter 106 interconnects bus 113 with an outside network 116 enabling data processing system 100 to communicate with other such systems. A screen (e.g., a display monitor) 115 is connected to system bus 113 by display adaptor 112, which may include a graphics adapter to improve the performance of graphics intensive applications and a video controller. In one embodiment, adapters 107, 106, and 112 may be connected to one or more I/O busses that are connected to system bus 113 via an intermediate bus bridge (not shown). Suitable I/O buses for connecting peripheral devices such as hard disk controllers, network adapters, and graphics adapters typically include common protocols, such as the Peripheral Component Interconnect (PCI). Additional input/output devices are shown as connected to system bus 113 via user interface adapter 108 and display adapter 112. A keyboard 109, mouse 110, and speaker 111 all interconnected to bus 113 via user interface adapter 108, which may include, for example, a Super I/O chip integrating multiple device adapters into a single integrated circuit.

In exemplary embodiments, the processing system 100 includes a graphics processing unit 130. Graphics processing unit 130 is a specialized electronic circuit designed to manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display. In general, graphics processing unit 130 is very efficient at manipulating computer graphics and image processing, and has a highly parallel structure that makes it more effective

13

than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel.

Thus, as configured in FIG. 9, the system 100 includes processing capability in the form of processors 101, storage capability including system memory 114 and mass storage 104, input means such as keyboard 109 and mouse 110, and output capability including speaker 111 and display 115. In one embodiment, a portion of system memory 114 and mass storage 104 collectively store an operating system to coordinate the functions of the various components shown in FIG. 9.

A number of implementations have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the disclosure. For example, various forms of the flows shown above may be used, with steps re-ordered, added, or removed. Accordingly, other implementations are within the scope of the following claims.

Embodiments and all of the functional operations described in this specification may be implemented in digital electronic circuitry, or in computer software, firmware, or hardware, including the structures disclosed in this specification and their structural equivalents, or in combinations of one or more of them. Embodiments may be implemented as one or more computer program products, i.e., one or more modules of computer program instructions encoded on a computer readable medium for execution by, or to control the operation of, data processing apparatus. The computer readable medium may be a machine-readable storage device, a machine-readable storage substrate, a memory device, a composition of matter effecting a machine-readable propagated signal, or a combination of one or more of them. The term "data processing apparatus" encompasses all apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus may include, in addition to hardware, code that creates an execution environment for the computer program in question, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an operating system, or a combination of one or more of them. A propagated signal is an artificially generated signal, e.g., a machine-generated electrical, optical, or electromagnetic signal that is generated to encode information for transmission to suitable receiver apparatus.

A computer program (also known as a program, software, software application, script, or code) may be written in any form of programming language, including compiled or interpreted languages, and it may be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program does not necessarily correspond to a file in a file system. A program may be stored in a portion of a file that holds other programs or data (e.g., one or more scripts stored in a markup language document), in a single file dedicated to the program in question, or in multiple coordinated files (e.g., files that store one or more modules, sub programs, or portions of code). A computer program may be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a communication network.

The processes and logic flows described in this specification may be performed by one or more programmable processors executing one or more computer programs to perform functions by operating on input data and generating output. The processes and logic flows may also be per-

14

formed by, and apparatus may also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit).

Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read only memory or a random access memory or both.

The essential elements of a computer are a processor for performing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or optical disks. However, a computer need not have such devices. Moreover, a computer may be embedded in another device, e.g., a tablet computer, a mobile telephone, a personal digital assistant (PDA), a mobile audio player, a Global Positioning System (GPS) receiver, to name just a few. Computer readable media suitable for storing computer program instructions and data include all forms of non-volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto optical disks; and CD ROM and DVD-ROM disks. The processor and the memory may be supplemented by, or incorporated in, special purpose logic circuitry.

To provide for interaction with a user, embodiments may be implemented on a computer having a display device, e.g., a CRT (cathode ray tube) or LCD (liquid crystal display) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user may provide input to the computer. Other kinds of devices may be used to provide for interaction with a user as well; for example, feedback provided to the user may be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user may be received in any form, including acoustic, speech, or tactile input.

Embodiments may be implemented in a computing system that includes a back end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front end component, e.g., a client computer having a graphical user interface or a Web browser through which a user may interact with an implementation, or any combination of one or more such back end, middleware, or front end components. The components of the system may be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network ("LAN") and a wide area network ("WAN"), e.g., the Internet.

The computing system may include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.

While this specification contains many specifics, these should not be construed as limitations on the scope of the disclosure or of what may be claimed, but rather as descriptions of features specific to particular embodiments. Certain features that are described in this specification in the context of separate embodiments may also be implemented in com-

15

ination in a single embodiment. Conversely, various features that are described in the context of a single embodiment may also be implemented in multiple embodiments separately or in any suitable subcombination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination may in some cases be excised from the combination, and the claimed combination may be directed to a subcombination or variation of a subcombination.

Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system components in the embodiments described above should not be understood as requiring such separation in all embodiments, and it should be understood that the described program components and systems may generally be integrated together in a single software product or packaged into multiple software products.

In each instance where an HTML file is mentioned, other file types or formats may be substituted. For instance, an HTML file may be replaced by an XML, JSON, plain text, or other types of files. Moreover, where a table or hash table is mentioned, other data structures (such as spreadsheets, relational databases, or structured files) may be used.

While the invention has been described with reference to some embodiments, it will be understood by those skilled in the art that various changes may be made and equivalents may be substituted for elements thereof without departing from the scope of the invention. In addition, many modifications may be made to adapt a particular situation or material to the teachings of the invention without departing from essential scope thereof. Therefore, it is intended that the invention not be limited to the particular embodiments disclosed as the best mode contemplated for carrying out this invention, but that the invention will include all embodiments falling within the scope of the appended claims.

What is claimed is:

1. A system for updating a graph-based data structure comprising a plurality of nodes and one or more edges connecting the plurality of nodes, the system comprising:
 - a processor that hosts an algorithm to incrementally update underlying data and abstractions of the graph-based data structure based on popular user selections;
 - a user interface that is in two-way operative communication with the processor, where the user interface is operative to present queries to a user as the user traverses the graph-based data structure; and
 - a memory database that is in communication with the processor and is adapted to receive, store, and recall data to/from the processor,
 where the processor is configured to execute the algorithm to:
 - cause the user interface to present the queries to the user;
 - determine patterns of access of the graph-based data structure by the user based at least in part on responses received from the user to the queries; and
 - update the graph-based data structure by converting a node of the plurality of nodes to an edge based at least in part on the patterns of access.

16

2. The system of claim 1, where queries presented to the user are dependent upon the amount of use of the queries made by preceding users.

3. The system of claim 1, where the algorithm develops additional queries around a subset of the queries that are used above a certain threshold level N_1 .

4. The system of claim 3, where the additional queries are beta-tested before being selected for further development.

5. The system of claim 4, where the further development comprises developing additional graph-based data structures for presentation to the user.

6. The system of claim 3, where the subset is a first subset, and where the algorithm develops additional graph-based data structures for presentation to the user around a second subset or the queries that are used above a certain threshold level N_2 , where N_2 is greater than N_1 .

7. The system of claim 3, where the subset is first subset, and where the algorithm stores a second subset of the queries that are used at a value of x in the database, where x is lower than N_1 .

8. The system of claim 1, where the processor is further configured to execute the algorithm to update the graph-based data structure by eliminating at least one of the plurality of nodes or at least one of the one or more edges from the graph-based data structure based on the patterns of access by the user.

9. The system of claim 1, where the patterns of access are generated by the user's use of pivots and data filtering.

10. The system of claim 1, where the processor is further configured to execute the algorithm to update the graph-based data structure based on queries submitted by the user.

11. The system of claim 1, where the memory works cooperatively with the processor to develop new queries, abstractions and graph-based data structures.

12. A method comprising:
traversing, via a user interface of a system, a graph-based data structure comprising a plurality of nodes and one or more edges connecting the plurality of nodes, the system comprising:

- a processor that hosts an algorithm to incrementally update underlying data and abstractions for the graph-based data structure based on popular user selections, where the user interface is in two-way operative communication with the processor, and where the user interface is operative to present queries to a user as the user traverses the graph-based data structure; and

- a memory database that is in communication with the processor and is adapted to receive, store, and recall data to/from the processor; and

- executing, by the processor, the algorithm, where executing the algorithm comprises:

- causing the user interface to present the queries to the user during the traversing;

- determining patterns of access of the graph-based data structure by the user based at least in part on responses received from the user to the queries; and
- updating the graph-based data structure by converting a node of the plurality of nodes to an edge based at least in part on the patterns of access.

13. The method of claim 12, where executing the algorithm further comprises developing additional queries around a subset of the queries that are used above a certain threshold level N_1 .

14. The method of claim 12, where executing the algorithm further comprises updating the graph-based data structure by eliminating at least one of the plurality of nodes or

17

at least one of the one or more edges from the graph-based data structure based on the patterns of access by the user.

15. A computer program product comprising:

a non-transitory storage medium readable by a processing circuit and storing instructions for execution by the processing circuit for performing a method comprising: traversing, via a user interface of a system, a graph-based data structure comprising a plurality of nodes and one or more edges connecting the plurality of nodes, the system comprising:

a processor that hosts an algorithm to incrementally update underlying data and abstractions of the graph-based data structure based on popular user selections,

where the user interface is in two-way operative communication with the processor, and where the user interface is operative to present queries to a user as the user traverses the graph-based data structure; and

a memory database that is in communication with the processor and is adapted to receive, store, and recall data to the processor; and

executing, by the processor, the algorithm, where executing the algorithm comprises:

causing the user interface to present the queries to the user during the traversing;

18

determining patterns of access of the graph-based data structure by the user based at least in part on responses received from the user to the queries; and updating the graph-based data structure by converting a node of the plurality of nodes to an edge based at least in part on the patterns of access.

16. The computer program product of claim **15**, where executing the algorithm further comprises developing additional queries around a subset of the queries that are used above a certain threshold level N_1 .

17. The computer program product of claim **16**, where the subset is a first subset, and where executing the algorithm further comprises developing additional graph-based data structures for presentation to the user around a second subset of the queries that are used above a certain threshold level N_2 , where N_2 is greater than N_1 .

18. The computer program product of claim **15**, where executing the algorithm further comprises updating the graph-based data structure by eliminating at least one of the plurality of nodes or at least one of the one or more edges from the graph-based data structure based on the patterns of access by the user.

* * * * *