US 20220035880A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2022/0035880 A1**

Miller et al. (43) **Pub. Date: Feb. 3, 2022**

(54) **DETERMINING FEASIBLE ITINERARY SOLUTIONS**

(71) Applicant: **Amgine Technologies (US), Inc.**, Dover, DE (US)

(72) Inventors: **Naomi Liora Miller**, New York City, NY (US); **Harold Roy Miller**, Toronto (CA); **Warren Stableford**, Burlington (CA)

(73) Assignee: **Amgine Technologies (US), Inc.**

(21) Appl. No.: **17/502,625**

(22) Filed: **Oct. 15, 2021**

Related U.S. Application Data

(63) Continuation of application No. 16/275,133, filed on Feb. 13, 2019, which is a continuation of application No. 15/595,795, filed on May 15, 2017, now Pat. No. 10,210,270, which is a continuation of application No. 15/069,791, filed on Mar. 14, 2016, now Pat. No. 9,659,099, which is a continuation-in-part of application No. 13/419,989, filed on Mar. 14, 2012, now Pat. No. 9,286,629.

(60) Provisional application No. 61/452,633, filed on Mar. 14, 2011.

Publication Classification

(51) **Int. Cl.**

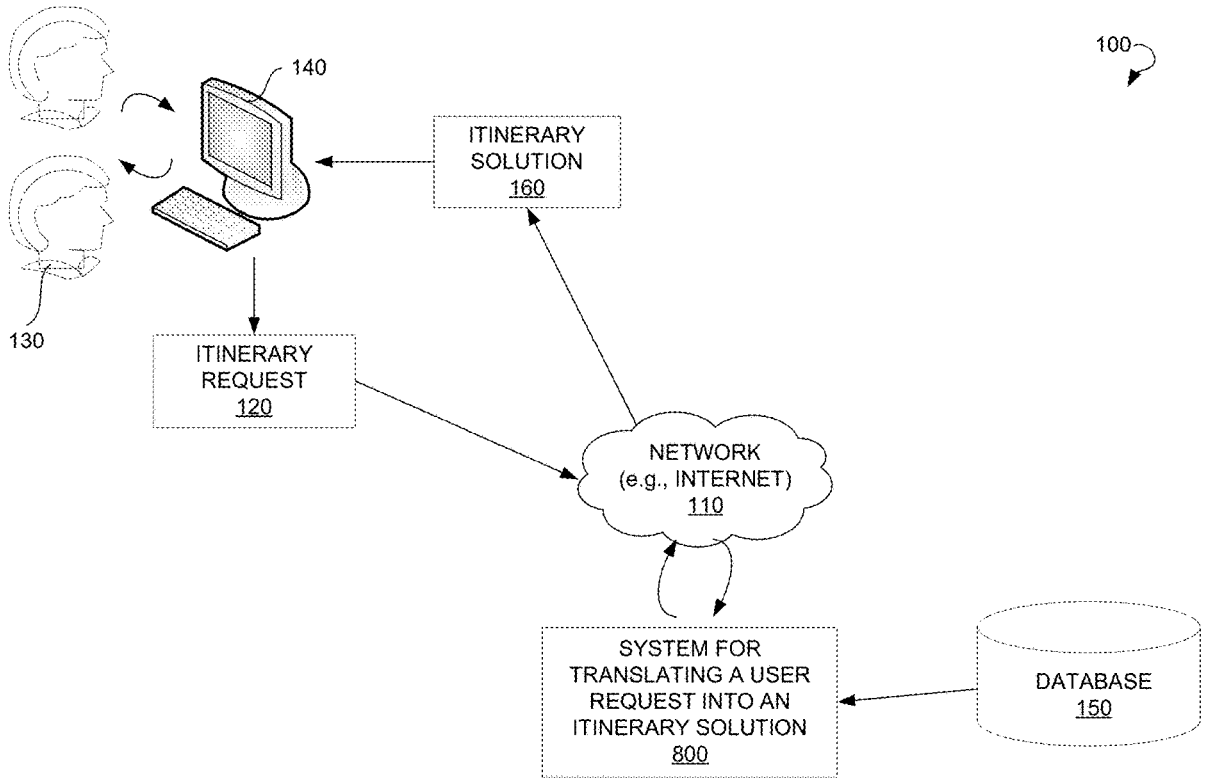
G06F 16/9535	(2006.01)
G06F 16/242	(2006.01)
G06Q 10/02	(2006.01)
G06F 40/30	(2006.01)

(52) **U.S. Cl.**

CPC ***G06F 16/9535*** (2019.01); ***G06F 40/30*** (2020.01); ***G06Q 10/02*** (2013.01); ***G06F 16/243*** (2019.01)

(57) **ABSTRACT**

A method for changing an itinerary based on a user change request is disclosed. The method may commence with receiving an itinerary request associated with one or more passengers. The method may continue with receiving a user itinerary change request associated with the itinerary network. The method may continue with generating an itinerary object associated with the user itinerary change request. The method may continue with modifying the itinerary network based on the itinerary object. The method may continue with processing the itinerary network using a topology of the itinerary network to create a plurality of tuples, the plurality of tuples including at least flight tuples and hotel tuples. The method may continue with performing a content search for the plurality of tuples for each of the one or more passengers. The method may continue with generating feasible itinerary solutions based on results of the content searches.



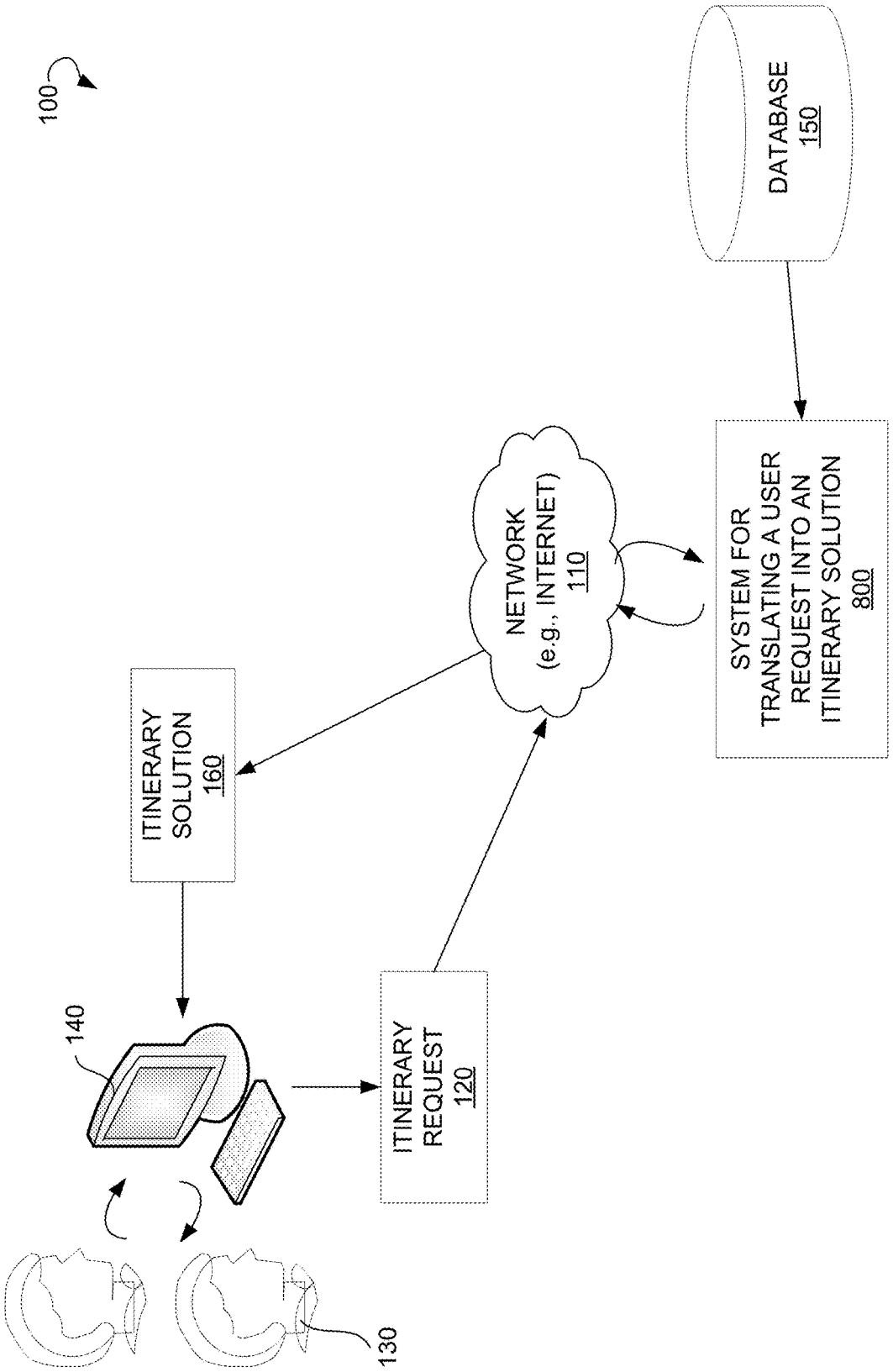

100
DATABASE
150
NETWORK
(e.g., INTERNET)
110
SYSTEM FOR
TRANSLATING A USER
REQUEST INTO AN
ITINERARY SOLUTION
800
ITINERARY
SOLUTION
160
ITINERARY
REQUEST
120
140
130

FIG. 1

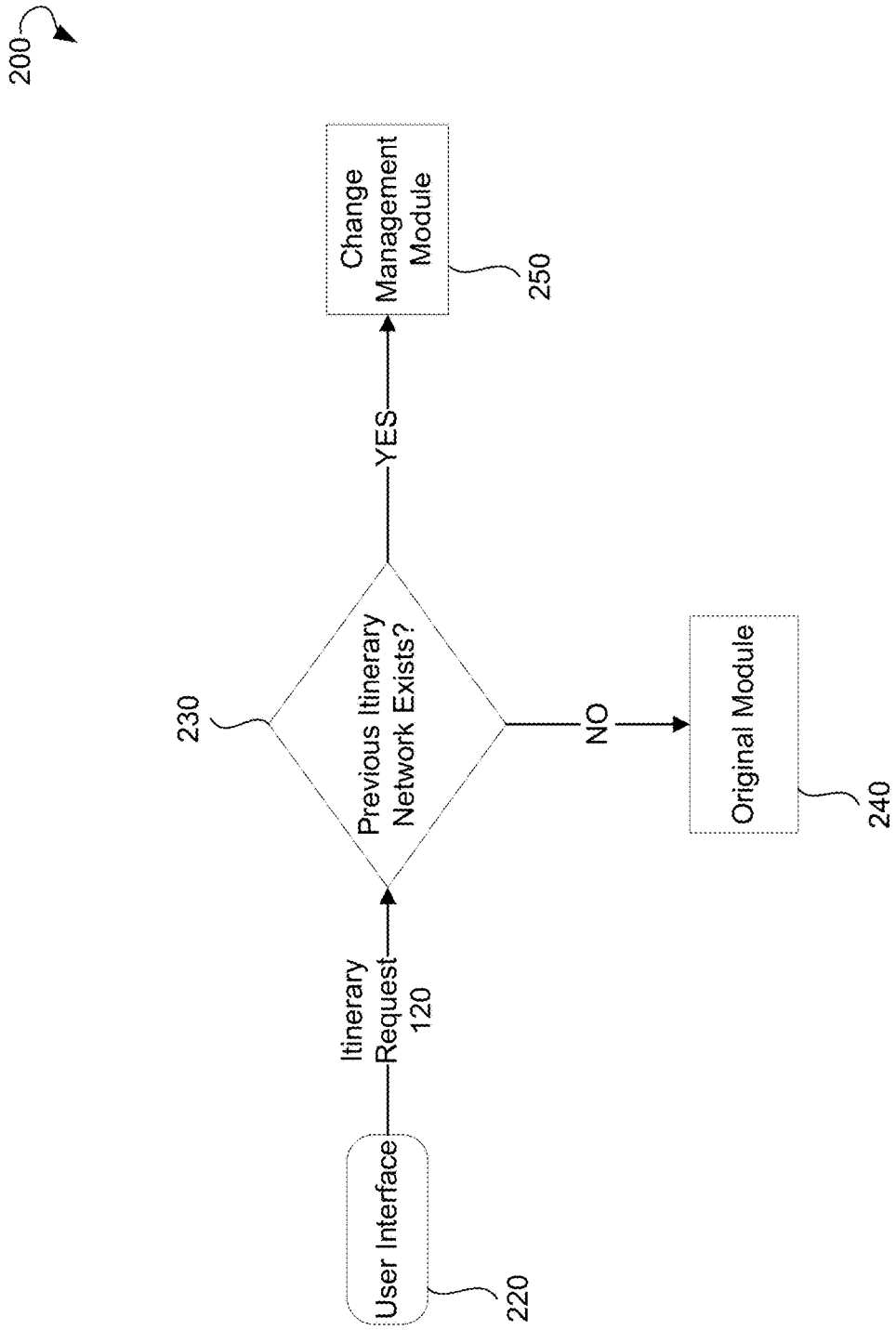

200
User Interface
220
Itinerary Request 120
Previous Itinerary Network Exists?
230
NO
Original Module
240
YES
Change Management Module
250

FIG. 2

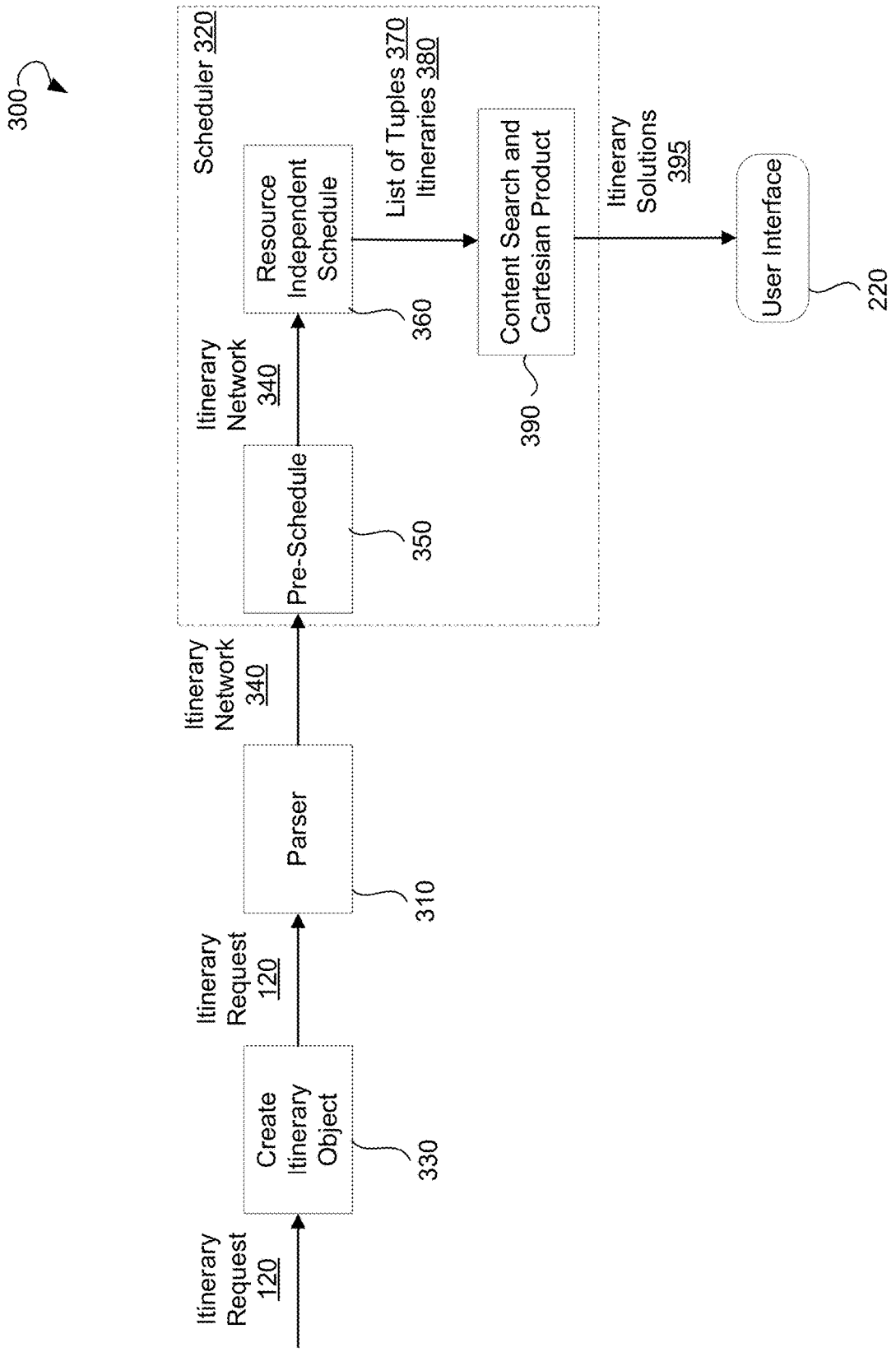

300
Itinerary Request 120
Create Itinerary Object
330
Itinerary Request 120
Parser
310
Itinerary Network 340
Pre-Schedule
350
Itinerary Network 340
Scheduler 320
Resource Independent Schedule
360
List of Tuples 370 Itineraries 380
Content Search and Cartesian Product
390
Itinerary Solutions 395
User Interface
220

FIG. 3

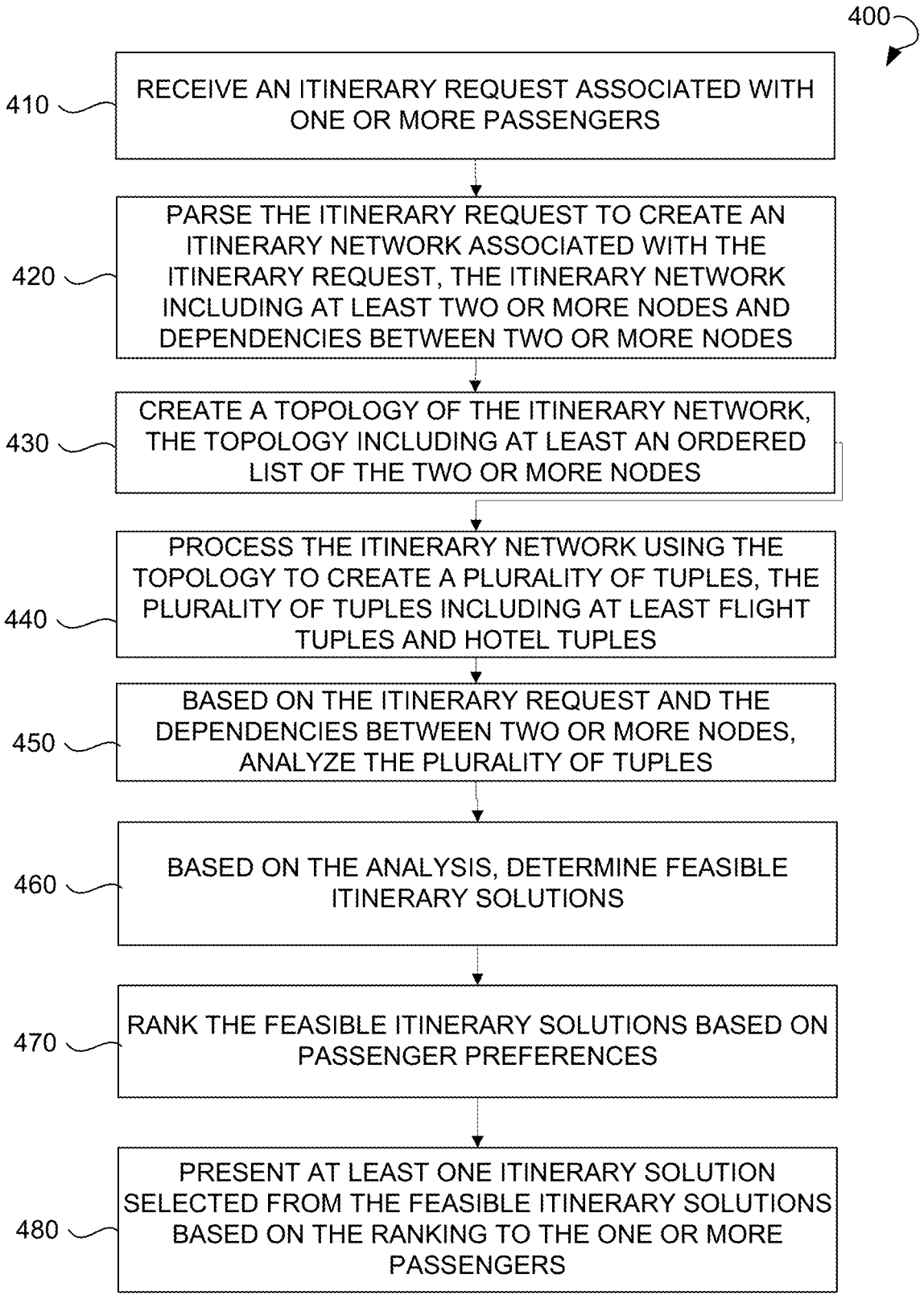

400
410
RECEIVE AN ITINERARY REQUEST ASSOCIATED WITH ONE OR MORE PASSENGERS
420
PARSE THE ITINERARY REQUEST TO CREATE AN ITINERARY NETWORK ASSOCIATED WITH THE ITINERARY REQUEST, THE ITINERARY NETWORK INCLUDING AT LEAST TWO OR MORE NODES AND DEPENDENCIES BETWEEN TWO OR MORE NODES
430
CREATE A TOPOLOGY OF THE ITINERARY NETWORK, THE TOPOLOGY INCLUDING AT LEAST AN ORDERED LIST OF THE TWO OR MORE NODES
440
PROCESS THE ITINERARY NETWORK USING THE TOPOLOGY TO CREATE A PLURALITY OF TUPLES, THE PLURALITY OF TUPLES INCLUDING AT LEAST FLIGHT TUPLES AND HOTEL TUPLES
450
BASED ON THE ITINERARY REQUEST AND THE DEPENDENCIES BETWEEN TWO OR MORE NODES, ANALYZE THE PLURALITY OF TUPLES
460
BASED ON THE ANALYSIS, DETERMINE FEASIBLE ITINERARY SOLUTIONS
470
RANK THE FEASIBLE ITINERARY SOLUTIONS BASED ON PASSENGER PREFERENCES
480
PRESENT AT LEAST ONE ITINERARY SOLUTION SELECTED FROM THE FEASIBLE ITINERARY SOLUTIONS BASED ON THE RANKING TO THE ONE OR MORE PASSENGERS

FIG. 4

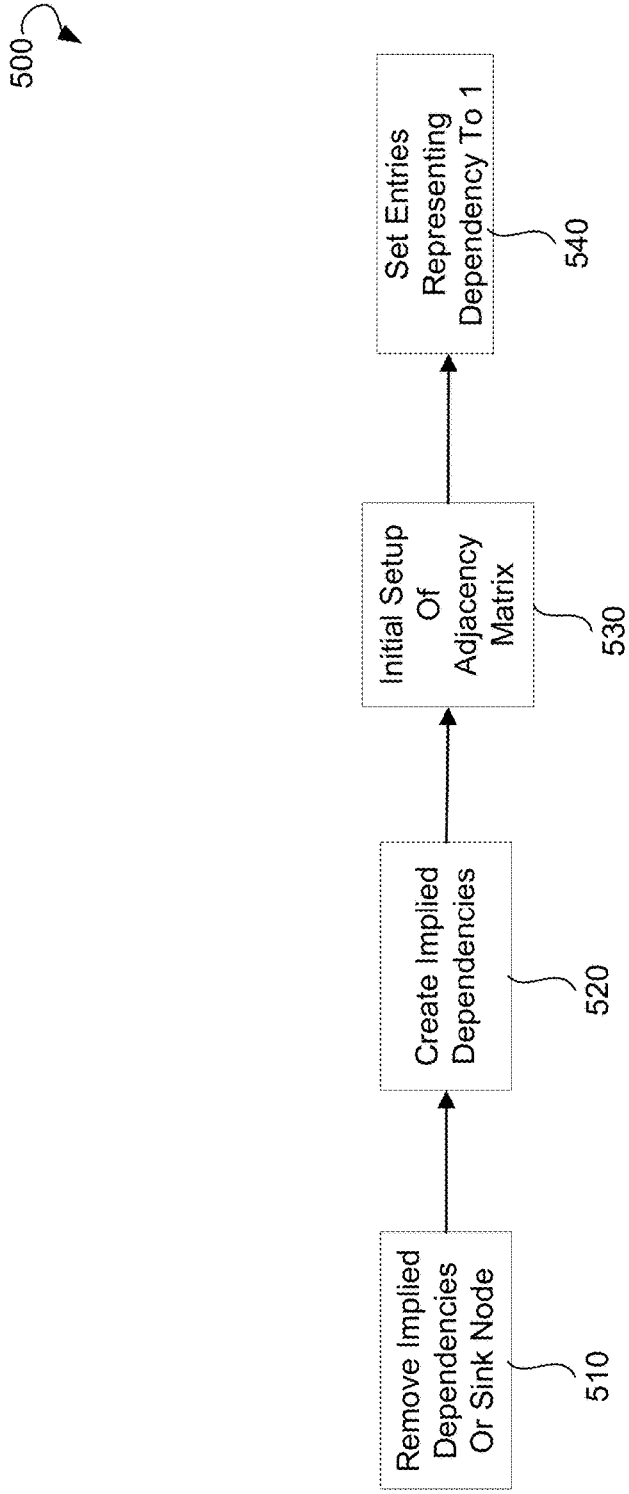

500
Remove Implied Dependencies Or Sink Node
510
Create Implied Dependencies
520
Initial Setup Of Adjacency Matrix
530
Set Entries Representing Dependency To 1
540

FIG. 5

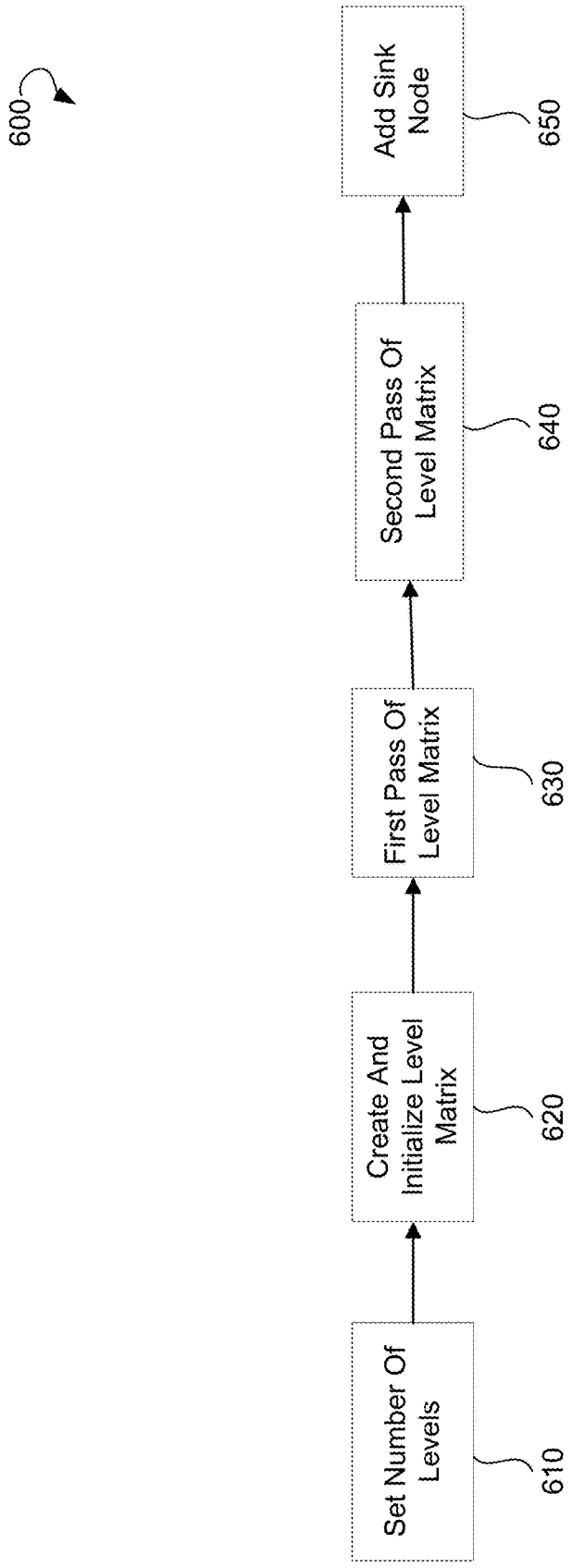

600
Set Number Of Levels
610
Create And Initialize Level Matrix
620
First Pass Of Level Matrix
630
Second Pass Of Level Matrix
640
Add Sink Node
650

FIG. 6

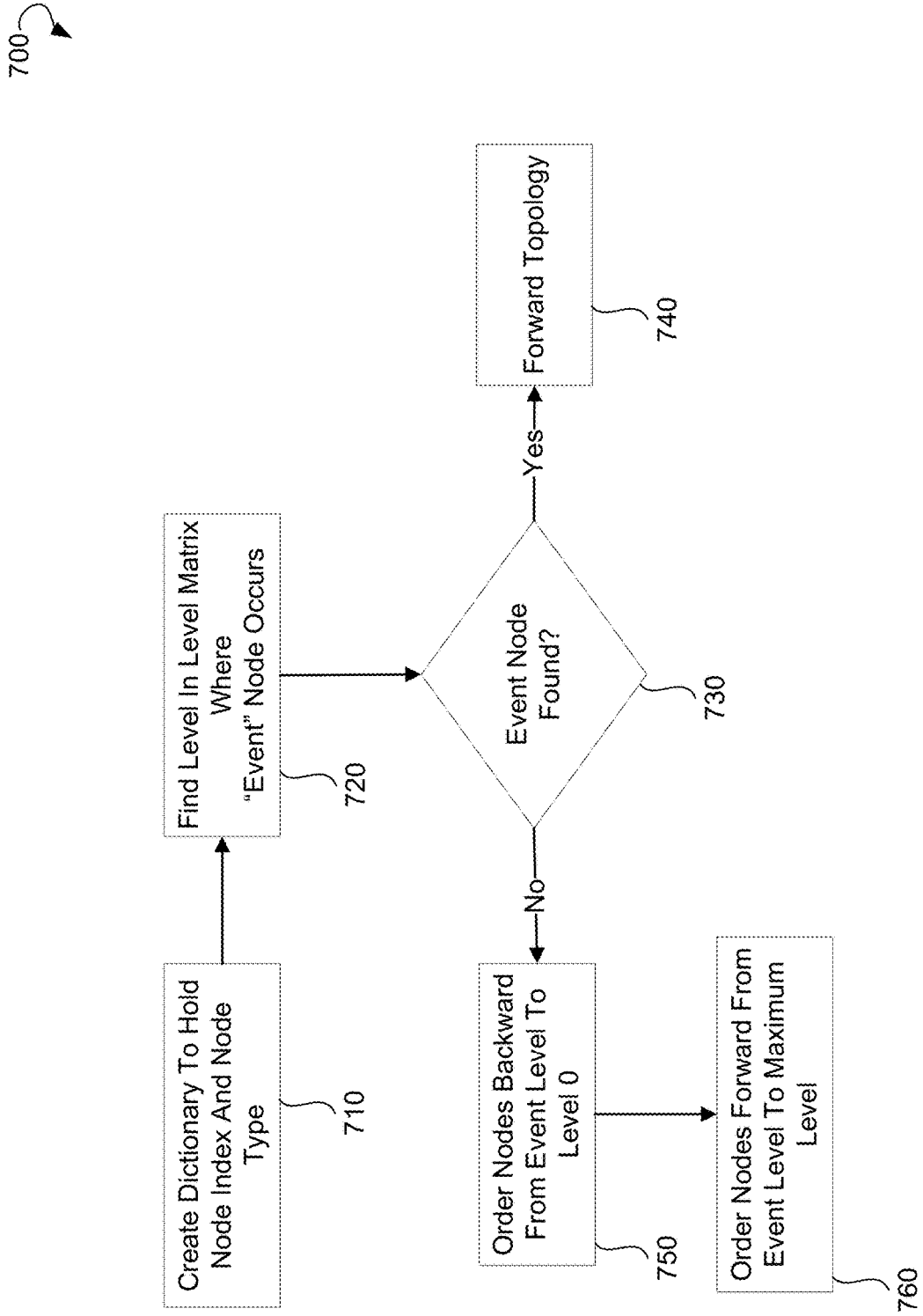

700
Create Dictionary To Hold Node Index And Node Type
710
Find Level In Level Matrix Where "Event" Node Occurs
720
Event Node Found?
730
Yes
Forward Topology
740
No
Order Nodes Backward From Event Level To Level 0
750
Order Nodes Forward From Event Level To Maximum Level
760

FIG. 7

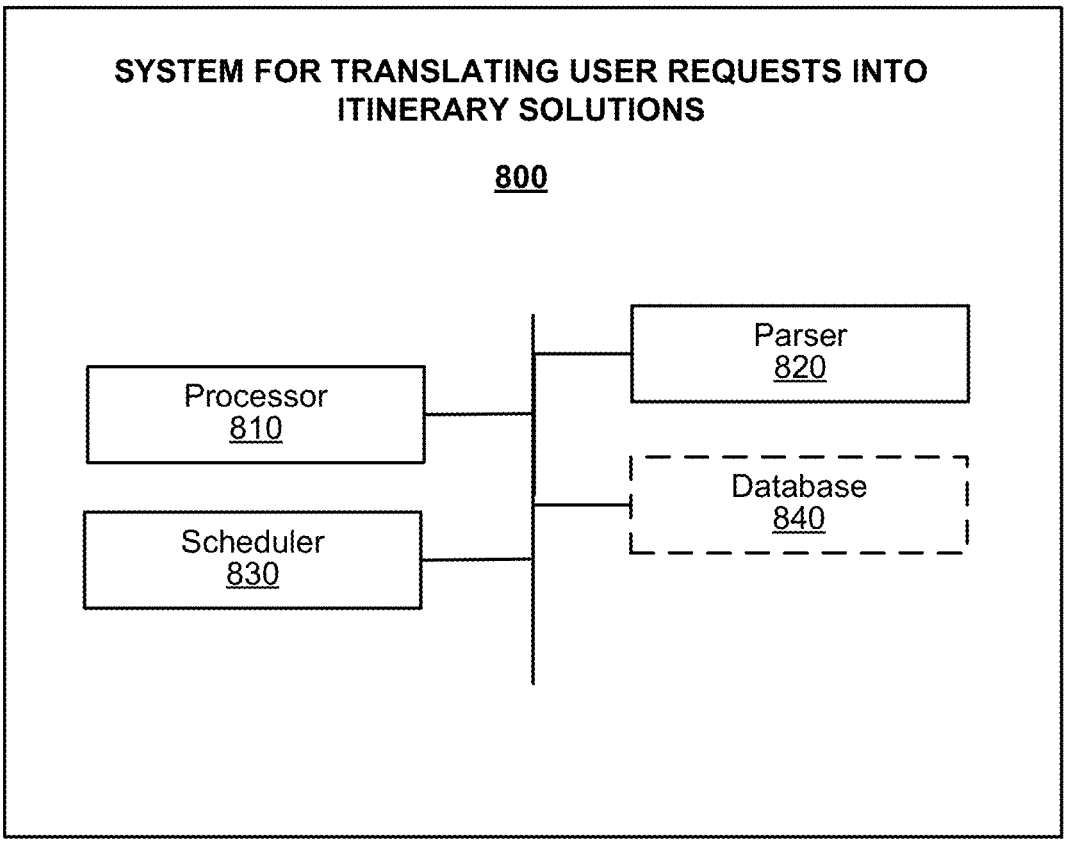

SYSTEM FOR TRANSLATING USER REQUESTS INTO ITINERARY SOLUTIONS
800
Processor
810
Scheduler
830
Parser
820
Database
840

FIG. 8

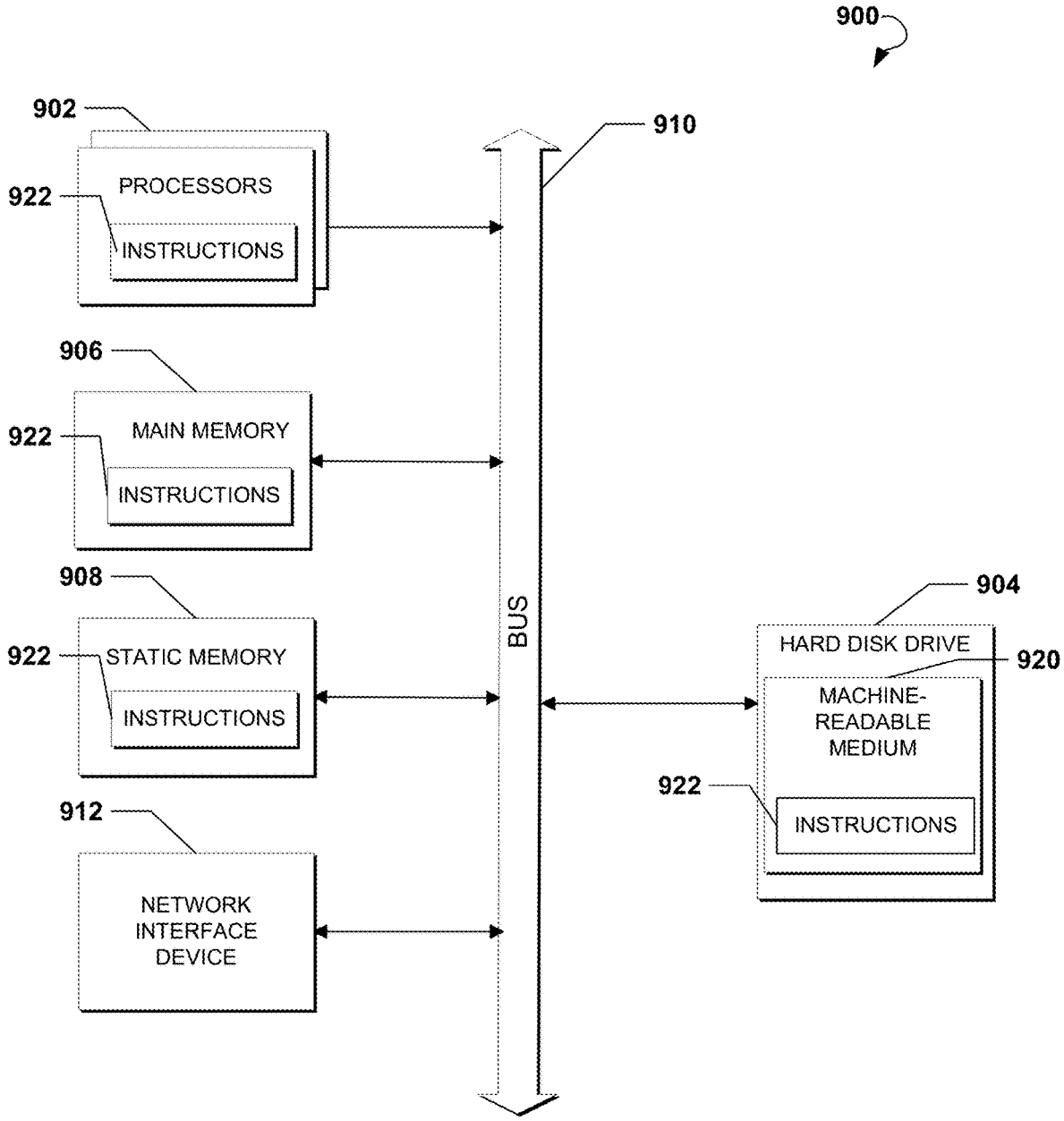

900
902
PROCESSORS
922
INSTRUCTIONS
910
906
MAIN MEMORY
922
INSTRUCTIONS
908
STATIC MEMORY
922
INSTRUCTIONS
BUS
904
HARD DISK DRIVE
920
MACHINE-READABLE MEDIUM
922
INSTRUCTIONS
912
NETWORK INTERFACE DEVICE

FIG. 9

DETERMINING FEASIBLE ITINERARY SOLUTIONS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application is a Continuation of U.S. patent application Ser. No. 16/275,133 filed Feb. 13, 2019, which is a Continuation of U.S. patent application Ser. No. 15/595,795 filed on May 15, 2017 and issued on Feb. 19, 2019 as U.S. Pat. No. 10,210,270, which is a Continuation of U.S. patent application Ser. No. 15/069,791 filed on Mar. 14, 2016 and issued on May 23, 2017 as U.S. Pat. No. 9,659,099, which is a Continuation-in-Part of U.S. patent application Ser. No. 13/419,989 filed on Mar. 14, 2012 and issued on Mar. 15, 2016 as U.S. Pat. No. 9,286,629, which claims the priority benefit of U.S. Provisional Patent Application No. 61/452,633, filed on Mar. 14, 2011. The disclosures of all of the above applications are incorporated by reference for all purposes.

TECHNICAL FIELD

[0002] The present disclosure relates to data processing and, more specifically, to translating user requests into itinerary solutions.

BACKGROUND

[0003] When searching for and booking flights and hotels, a travel consumer may utilize choice environments provided by online travel agencies and websites. These choice environments can provide travel-related information and advice on everything from destinations to hotels, related points of interest, and pricing data for a vast array of goods and services. However, a choice task of the travel consumer is often encumbered by information abundance and by legacy technology platforms that almost invariably complicate the consumer choice problem.

[0004] Moreover, travel agents and online travel agencies may need to sort through a plurality of records and manually select various options when selecting and scheduling the travel itinerary for the travel consumer. This process becomes particularly difficult when multiple databases are involved in developing complex itineraries, such as reserving multiple flights, hotels, cars, and restaurants, and for developing itineraries for groups of travel consumers. Furthermore, travel agents and online travel agencies may charge travel consumers for making changes to their itineraries and these changes can take from hours to days to take effect.

[0005] Additionally, a conventional travel reservation process does not typically involve taking into consideration previous travel itineraries associated with the travel consumer. Thus, searching for and selecting a travel itinerary is done without analysis of selections or preferences associated with previous requests.

SUMMARY

[0006] This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

[0007] According to one example embodiment of the disclosure, a system for translating user requests into itinerary solutions is provided. The system may include a processor, a parser in communication with the processor, and a scheduler in communication with the processor. The processor may be operable to receive an itinerary request associated with one or more passengers. Furthermore, the processor may be operable to present at least one itinerary solution to the one or more passengers. The at least one itinerary solution may be selected from feasible itinerary solutions based on ranking.

[0008] The parser may be operable to parse the itinerary request to create an itinerary network associated with the itinerary request. The itinerary network may include at least two or more nodes and dependencies between two or more nodes. The scheduler may be operable to create a topology of the itinerary network. The topology may include at least an ordered list of the two or more nodes. The scheduler may be further operable to process the itinerary network using the topology to create a plurality of tuples, where a single node may correspond to a single tuple. The plurality of tuples may include at least flight tuples and hotel tuples. The scheduler may be operable to analyze the plurality of tuples based on the itinerary request and the dependencies between two or more nodes. Based on the analysis, the scheduler may determine the feasible itinerary solutions. Furthermore, the scheduler may be operable to rank the feasible itinerary solutions based on at least passenger preferences.

[0009] According to another example embodiment of the disclosure, a method for translating user requests into itinerary solutions is provided. The method may commence with receiving an itinerary request associated with one or more passengers. The method may continue with parsing the itinerary request to create an itinerary network associated with the itinerary request. The itinerary network may include at least two or more nodes and dependencies between two or more nodes. Upon the parsing, a topology of the itinerary network may be created. The topology may include at least an ordered list of the two or more nodes. The method may further include processing the itinerary network using the topology to create a plurality of tuples, where a single node may correspond to a single tuple. The plurality of tuples may include at least flight tuples and hotel tuples. Furthermore, the plurality of tuples may be analyzed based on the itinerary request and the dependencies between two or more nodes. The method may continue with determining feasible itinerary solutions based on the analysis. The method may further include ranking the feasible itinerary solutions based on passenger preferences. Furthermore, the method may include presenting at least one itinerary solution to the one or more passengers. The at least one itinerary solution may be selected from the feasible itinerary solutions based on the ranking.

[0010] Other example embodiments of the disclosure and aspects will become apparent from the following description taken in conjunction with the following drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] Embodiments are illustrated by way of example and not limitation in the figures of the accompanying drawings, in which like references indicate similar elements.

[0012] FIG. **1** illustrates an environment within which systems and methods for translating user requests into itinerary solutions can be implemented.

[0013] FIG. **2** is a flow diagram showing a process for translating user requests into travel itineraries depending on a previous itinerary network.

[0014] FIG. **3** is a block diagram showing a translation of user requests into travel itineraries via a parser and a scheduler.

[0015] FIG. **4** is a process flow diagram showing a method for translating user requests into itinerary solutions.

[0016] FIG. **5** is block diagram illustrating a process for building an adjacency matrix.

[0017] FIG. **6** is a block diagram illustrating a process for building a level matrix.

[0018] FIG. **7** is a block diagram illustrating a process for creating a topology.

[0019] FIG. **8** is a block diagram showing various modules of a system for translating user requests into itinerary solutions.

[0020] FIG. **9** shows a diagrammatic representation of a computing device for a machine in the exemplary electronic form of a computer system, within which a set of instructions for causing the machine to perform any one or more of the methodologies discussed herein can be executed.

DETAILED DESCRIPTION

[0021] The following detailed description includes references to the accompanying drawings, which form a part of the detailed description. The drawings show illustrations in accordance with exemplary embodiments. These exemplary embodiments, which are also referred to herein as "examples," are described in enough detail to enable those skilled in the art to practice the present subject matter. The embodiments can be combined, other embodiments can be utilized, or structural, logical, and electrical changes can be made without departing from the scope of what is claimed. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope is defined by the appended claims and their equivalents.

[0022] The disclosure relates to systems and methods for translating user requests into itinerary solutions. More specifically, upon receiving a user request, such as an itinerary request associated with one or more passengers, the itinerary request is forwarded to a parser. The parser may be responsible for parsing the itinerary request to determine nodes associated with the itinerary request and dependencies between the nodes. The node may represent an attribute associated with the itinerary request, such as a city, a flight, a hotel, a date, time, and so forth. The nodes may be combined into an itinerary network.

End-to-End Process Overview

[0023] Based on the itinerary request, an itinerary object associated with the itinerary request may be created. The itinerary object, also referred to as an itinerary network object, is a set of data that contains at least one of the following: an itinerary request, an itinerary network created by the parser, structures created by the scheduler (such as an adjacency matrix, a level matrix, and a topology), lists of partitions, itinerary solutions, a passengers list, function calls, and other related items, such as a conversation message and the like. The itinerary network object may be passed to a pre-scheduling process, also referred herein to as a pre-schedule algorithm. Upon creation of the itinerary network, a scheduler may create a level matrix for the itinerary network by selecting main nodes and child nodes that depend on the main nodes. Furthermore, the scheduler may use the pre-schedule algorithm to create a level matrix of the itinerary network. Based on the level matrix, a topology of the itinerary network may be created. In the topology, the nodes are organized in a form of an ordered list.

[0024] Using the pre-schedule algorithm a level matrix of nodes and a topology of nodes are created. Based on the topology, the parser can analyze the itinerary network to create tuples associated with the itinerary request. For example, flight tuples and hotel tuples can be created. As used herein, a tuple is a set of objects associated with a node. In an example embodiment, in a topology, a single node may correspond to a single tuple. The parsers use a resource independent scheduling process, also referred to as a resource independent schedule algorithm, to process through the topology one node at a time to create a flight tuple and/or a hotel tuple for a given flight node and/or hotel node. Tuples may not be assigned to city nodes.

[0025] Each flight tuple and/or hotel tuple provides date intervals and time intervals to be used in a content search for an itinerary solution (i.e., a flight solution or a hotel solution) for the flight node or the hotel node. In the case of flight tuples, date intervals and/or time intervals may include widebucket intervals. In the case of hotel tuples, date intervals and/or time intervals may include check-in dates, check-in times, checkout dates and check-out times. Flight tuples have an additional optimal bucket interval used for content search and for ranking solutions at a later stage.

Partition

[0026] The flight tuples and hotel tuples may then be separated into groups based on the passenger. A partition subset may be determined based on a relation or a property of interest associated with flight tuples. For each passenger, one or more partitions of the flight tuples associated with that passenger may be created.

[0027] For each passenger, a single partition of the set of flight tuples into one or more disjoint groups is created, based on a relation or property of interest. The groups are called partition subsets.

Content Search

[0028] For each passenger, the following steps may be performed. The content search may be performed based on each hotel tuple, obeying the check-in and checkout dates and times specified in the hotel tuple. The itinerary solutions found based on the content search may be stored in a tuple solution bucket. Then, the content search may be performed separately on each partition subset of the single partition for the passenger. If the subset contains one flight, itinerary solutions for a one-way flight may be returned. If there are two flights in the subset, a priced combination of two flights may be returned. The widebucket date and time intervals from the flight tuples may be obeyed in each search. The solutions may be stored in a partition subset solution bucket.

[0029] In parallel, the content search can be performed across all partition subsets for all passengers. For each partition subset, the flight and hotel attributes for the content search may be provided by the tuple time buckets. A set of flight solutions and hotel solutions in compliance with restrictions associated with the itinerary request, such as

time restrictions, may be returned and stored for each of the partition subsets. If the only restriction is time, the buckets within the solution fall. A vital aspect of the scheduling is that after the ranking, the scheduling provides the best fitted itinerary solutions for the restrictions placed on the itinerary request. It can be difficult to find results if each time the content search is made and only results that exactly match the restrictions are found. Multi-objective ranking is the mechanism by which providing of the best fitted itinerary solutions is achieved.

Ranking

[0030] Ranking of the flight solutions and hotel solutions is performed based on the results of the content search before the flight solutions and hotel solutions are returned to the passenger. Each flight solution and hotel solution is scored against multiple user preferences.

Cartesian Product

[0031] For each passenger, the Cartesian product algorithm may create feasible itinerary solutions by combining solutions from the partition subsets and individual hotel tuples correctly into one set. These combined solutions may be ranked.

[0032] A solution space for the itinerary is formed by taking the Cartesian product of the sets of solutions for each of the partition subsets. Infeasible combinations of solutions may be removed from a solution list. Furthermore, solutions from the partition subsets and individual hotel tuples can be combined into one set of feasible itinerary solutions and stored in the solution list in the partition for each passenger. The solutions can be ranked and a predetermined number of top solutions can be returned and provided to the passengers via a user interface. It another example embodiment, more than one passenger may be combined into a partition subset.

[0033] If same flight dependencies and hotel dependencies exist for different passengers, an itinerary wide Cartesian product algorithm may be performed. Itinerary wide Cartesian product algorithm may combine itinerary solutions from each passenger and rank the itinerary solutions obeying same flight dependencies hotel dependencies, if any exist. The highest ranked itinerary solution may be returned.

[0034] Referring to the drawings, FIG. **1** illustrates an environment **100** within which the systems and methods for translating user requests into itinerary solutions can be implemented, in accordance with some embodiments. An itinerary request **120** associated with one or more passengers **130** may be received, for example, via a user interface displayed on a user device **140**. In particular, the user device **140**, in some example embodiments, may include a Graphical User Interface (GUI) for displaying the user interface associated with a system **800** for translating user requests into itinerary solutions. In a typical GUI, instead of offering only text menus or requiring typed commands, the system **800** may present graphical icons, visual indicators, or special graphical elements called widgets that may be utilized to allow the passengers **130** to interact with the system **800**.

[0035] The itinerary request **120** may be transmitted to the system **800** for translating user requests into itinerary solutions via a network **110**. The network **110** may include the Internet or any other network capable of communicating data between devices. Suitable networks may include or interface with any one or more of, for instance, a local intranet, a Personal Area Network, a Local Area Network (LAN), a Wide Area Network (WAN), a Metropolitan Area Network, a virtual private network, a storage area network, a frame relay connection, an Advanced Intelligent Network connection, a synchronous optical network connection, a digital T1, T3, E1 or E3 line, Digital Data Service connection, Digital Subscriber Line connection, an Ethernet connection, an Integrated Services Digital Network line, a dial-up port such as a V.90, V.34 or V.34bis analog modem connection, a cable modem, an Asynchronous Transfer Mode connection, or an Fiber Distributed Data Interface or Copper Distributed Data Interface connection. Furthermore, communications may also include links to any of a variety of wireless networks, including Wireless Application Protocol, General Packet Radio Service, Global System for Mobile Communication, Code Division Multiple Access or Time Division Multiple Access, cellular phone networks, Global Positioning System, cellular digital packet data, Research in Motion, Limited duplex paging network, Bluetooth radio, or an IEEE 802.11-based radio frequency network. The network **110** can further include or interface with any one or more of an RS-232 serial connection, an IEEE-1394 (Firewire) connection, a Fiber Channel connection, an IrDA (infrared) port, a SCSI (Small Computer Systems Interface) connection, a Universal Serial Bus (USB) connection or other wired or wireless, digital or analog interface or connection, mesh or Digi® networking. The network **110** may be a network of data processing nodes that are interconnected for the purpose of data communication. The network **110** may include any suitable number and type of devices (e.g., routers and switches) for forwarding commands, content, and/or web object requests from each client to the online community application and responses back to the clients.

[0036] The user device **140** may include a mobile telephone, a personal computer (PC), a laptop, a smart phone, a tablet PC, and so forth. The system **800** may be a server-based distributed application, thus it may include a central component residing on a server and one or more client applications residing on one or more user devices and communicating with the central component via the network **110**. The passengers **130** may communicate with the system **800** via a client application available through the user device **140**.

[0037] The itinerary request **120** may be analyzed based on travel itineraries available from the optional database **150** with reference to preferences of the passengers **130**. An itinerary solution **160** suiting the itinerary request **120** and preferences of the passengers **130** may be determined. The itinerary solution **160** may be presented to the passengers **130** by displaying the itinerary solution **160** via the user interface on a screen of the user device **140**.

[0038] FIG. **2** is a flow diagram **200** showing an overall process for translating user requests into travel itineraries depending on a previous itinerary network. More specifically, an itinerary request **120** associated with one or more passengers may be received via a user interface **220**. The itinerary request **120** may be processed by an original module **240** or by a change management module **250** depending on whether one or more previous itinerary networks exist for these passengers, as determined at block **230**. In an example embodiment, differences between existing itinerary networks and new itinerary networks (if no existing

itinerary networks are found) may be resolved by performing an intelligent ranking process.

[0039] FIG. **3** is a schematic block diagram **300** showing translating user requests into travel itineraries by the original module within the environment described with reference to FIG. **1**, according to example embodiments. The translation of user requests into travel itineraries is performed by a processor (not shown), a parser **310**, a scheduler **320**, and a user interface **220**. When the processor receives an itinerary request **120** from one or more passengers, an itinerary object may be created at block **330**. The itinerary object is a set of data that contains at least the following data: the itinerary request **120**, an itinerary network **340** created by the parser **310**, structures created by the scheduler **320** (such as an adjacency matrix, a level matrix, and a topology), lists of partitions, itinerary solutions, a passengers list, function calls, and other related items, such as a conversation message and the like. The itinerary object may contain all the information needed for the scheduler **320** and solutions which the scheduler **320** generates. In an example embodiment, a file 'Itinerary.es' may be created to store the itinerary object. Additionally, in relation to command and control, the itinerary object may allow communication between the parser **310** and the scheduler **320**.

[0040] When the itinerary object is created, the itinerary request **120** and the itinerary object may be forwarded to the parser **310**. The parser **310** may create an itinerary network **340** and forward the itinerary network **340** to the scheduler **320**. The scheduler **320** analyzes the itinerary network **340**, individual passenger profiles, and availability of content. The main processes of the scheduler **320** are pre-scheduled at block **350**, followed by forwarding the itinerary network **340** to block **360** for resource-independent scheduling of the itinerary network **340**. Upon the resource-independent scheduling, a list **370** of tuples and itineraries **380** are provided to block **390**. The tuples may be separated into groups based on the passenger. A partition subset may be determined based on a relation or a property of interest associated with the tuples. For each passenger, one or more partitions of the flight tuples associated with that passenger may be created. A content search may be performed over the partition subset and a Cartesian product may be applied to results of the content search on the partition subsets. Itinerary solutions **395** feasible for the itinerary request **120** may be forwarded to the user interface **220**. Steps performed by the parser **310** and the scheduler **320** are described in more detail below with reference to FIGS. **4-7**.

[0041] FIG. **4** is a detailed process flow diagram showing a method **400** for translating user requests into travel itineraries, according to example embodiments. The method **400** may commence with receiving an itinerary request at operation **410**. In an example embodiment, the itinerary request may be provided via at least one of the following: a natural language, a typed text, a selection of preexisting options, and so forth. In a further example embodiment, the itinerary request may be associated with one passenger or a group of passengers. Furthermore, based on the receipt of the itinerary request, an itinerary object associated with the itinerary request may be created.

[0042] At step **420**, upon receipt of the itinerary request, the parser performs parsing of the itinerary request to create an itinerary network associated with the itinerary request. In an example embodiment, the itinerary network includes a node list, a passenger list, and a dependency list. The node list may contain at least two or more nodes. Each of the nodes represents an attribute, such as a city, a flight, a hotel, and so forth. The dependency list may contain dependencies between two or more nodes, in particular, a directed relation between two nodes, such as a level dependency. In an example embodiment, the node may have additional information concerning an itinerary object that the node represents, such as a passenger or a group of passengers and attributes requested according to the itinerary request, for example, a date, an airline, a hotel brand, and so forth.

[0043] The scheduler receives the itinerary network created by the parser. The scheduler is responsible for building of a set of itinerary solutions (e.g., flights, hotels, and so forth), taking into account the structure of the itinerary network, individual passenger profiles, and the availability in content. In particular, the scheduler performs pre-scheduling of the itinerary network, resource independent scheduling of the itinerary network, a content search, and applying a Cartesian product to results of the content search on the partition subsets.

A Pre-Scheduling Process of the Scheduler.

[0044] A pre-scheduling process includes creating an adjacency matrix, a level matrix, and a topology.

Adjacency Matrix.

[0045] Still referring to FIG. **4**, the method **400** may optionally include creating an adjacency matrix of the itinerary network based on the classification of the nodes into main nodes and child nodes. The adjacency matrix is an (n×n) matrix, where n is the number of nodes in the network. Each (i, j) entry has a value of (1) if there is a directed dependency between node i and node j in the network, otherwise, each (i, j) entry has value of (−1) or (0).

[0046] FIG. **5** is a block diagram **500** illustrating a process of building the adjacency matrix. More specifically, at block **510**, any implied dependency provided by the parser or a sink node can be removed. At block **520**, implied dependencies between the nodes may be created. More specifically, the scheduler may classify each of the nodes into a main node and a child node based on the dependencies between the nodes. The child node is dependent on the main node. Therefore, an implied dependency between a main node and its child node can be created. For example, an implied dependency between a city node being the main node and a hotel node being a child node may be created. The implied dependency may be also created between a flight node which is a main node and a hotel node which is a child node of a destination city. At block **530**, an initial setup of the adjacency matrix can be performed. In particular, dimensions (n+2×n+2) can be set, where n is a number of nodes in the itinerary network. The adjacency matrix can be created and all entries initially set to (−1). At block **540**, entries that represent dependencies output by the parser and implied dependencies are set to (1).

Level Matrix.

[0047] Referring again to FIG. **4**, the method **400** may optionally include organizing the nodes into levels to create a level matrix of the itinerary network. In other words, the level matrix, also referred to as a directed acyclic graph matrix, can organize nodes into levels. In an example embodiment, 'Level 0' of the level matrix includes node A

and node B, 'Level 1' includes node C, node D, and node E, and 'Level 2' includes node F. The nodes in each level have dependent nodes at the previous level. The nodes in 'Level 0' have no dependencies. The nodes at 'Level 1' have one or more dependent nodes at 'Level 0'. The nodes at 'Level 2' have one or more dependent nodes at 'Level 1', and so forth.

[0048] FIG. **6** is a block diagram **600** illustrating a process of building a level matrix. More specifically, at block **610**, a number of levels can be set. In particular, dimensions (n+1×n+1) can be set, where n is number of nodes in the network. The row index may represent the level, the column index may represent the order in which the node was put in. For example, an entry in (i, j) is the index of the node placed in level i in j^{th} place. At block **620**, the level matrix may be created and initialized. All entries may be initially set to (−1). At block **630**, a first pass of the level matrix may be performed. For this purpose, first, all nodes with no dependencies may be found and placed in level 0. Second, all nodes with dependencies at level 0 may be found and placed in level 1. Successive level nodes may be created until a maximum level is reached. At block **640**, a second pass of the level matrix may be performed. In particular, the scheduler may go back through the level matrix and remove duplicate nodes. The nodes in the level matrix may be ordered such that the nodes only appear once and at the latest stage or highest level. Since the level matrix is used for recusing through the nodes, level matrix may ensure that, when performing a forward pass, all dependent nodes for a specific node have already been scheduled, or in the case of the creating of the topology the ordering of the topology also may ensure this condition. At block **650**, a sink node may be added.

Topology.

[0049] Still referring to FIG. **4**, the method **400** may further include operation **430**, in which the scheduler may create a topology of the itinerary network. The level matrix provides the basis on which the topology of the itinerary network may be built. The topology is an ordered list of two or more nodes to be scheduled. FIG. **7** is a block diagram **700** illustrating a process for creating the topology. As shown on FIG. **7**, at block **710**, a dictionary may be created to hold a node index and a node type. At block **720**, a level in a level matrix where an event node occurs may be found. If the event node is found at a decision block **730**, a forward topology may be created at block **740**. In the forward topology, the scheduler may start at level 0 of the level matrix and add the nodes to the topology in the order the nodes occur. This process continues for levels 1 through the maximum level of the level matrix.

[0050] In an example embodiment, if level3 denotes the level in the level matrix being considered, k denotes the number of nodes reached at level3, and level3 is set to 0, the following example program code can be used to create the forward topology:

```
while (level3 < maxlevels) do
    Set k = 0
  while (k < levelMatrix.length) do
      Search for next node in level3
    If found add it to topology
    k++
    end
```

-continued

```
    level3++
  end.
```

[0051] If the event node is not found at the decision block **730**, an event topology may be created. The scheduler may start at level at which the event node is found and add all nodes at that level. Then, for each node at that level, the scheduler may find dependent nodes in the previous level, and add those dependent nodes to the topology. This process is continued backwards through the levels until level 0. Afterwards, the scheduler performs a pass of the forward topology from the first level after the event level. As shown on FIG. **7**, at block **750**, nodes may be ordered backward from the event level to level 0. At block **760**, the nodes may be ordered forward from the event level to the maximum level.

Resource Independent Scheduling Process.

[0052] Referring again to FIG. **4**, the method **400** may further include processing the itinerary network by the scheduler using the topology to create a plurality of tuples at operation **440**. In particular, in an example embodiment, the resource independent scheduling process, or a resource independent schedule algorithm, for each flight node, determines departure and arrival date and time windows, as a function of the itinerary request and the dependencies between two or more nodes in the itinerary network. The departure and arrival date and time windows are used for the content search and ranking of results of the content search. The resource independent scheduling process is also used to also determine check-in and check-out dates for each hotel node in the itinerary network.

[0053] The following example program code can be used to determine check-in and check-out dates:

```
ForEach (node n in the topology){
  If ( n is a flight node){
    Set WideBucket Departure and Arrival Windows
    Set OptimalBucket Departure and Arrival Windows
  }
  If ( n is a hotel node){
    Set check-in and check-out dates for the hotel
  }
  If ( n is a city node) {
  Do Nothing
```

[0054] The method **400** may further include analyzing the plurality of tuples by the scheduler based on the itinerary request and dependencies between two or more nodes at operation **450**. In particular, the resource independent scheduling process may include determining widebucket departure windows, widebucket arrival windows, and optimalbucket windows.

WideBucket Departure Window.

[0055] A WideBucket departure window may be used for determining departure windows for the content search. When the resource independent schedule algorithm hits a flight node, the resource independent schedule algorithm may looks for all dependent flight nodes of the flight node in the itinerary network. The dependent nodes may include previous flights of the passenger, or flights from other

passenger's itineraries which impact this passenger, for example if they these passengers fly together. The resource independent schedule algorithm then sets the departure window based on the requested date and time and the dependent flights. The resource independent schedule algorithm for determining the widebucket departure window is described in detail below.

[0056] Let F_i be a flight node. The departure window for F_i is $[x_{st}, x_{fi}]$ and the arrival window is $[d_{st}, d_{fi}]$. r_i is a requested date and t_i is requested time.

[0057] In the first step, the start and end of departure window are initialized to the current date and time (D). $[x_{st}, x_{fi}]=[D,D]$. In the second step, all dependent flight nodes for this flight node are found in the itinerary network. The example program code for these steps may be as follows:

```
foreach (node in nDependencies) do
  switch (NodeType of node) do
    case Flight
      (find a flight tuple corresponding to the node in a list and add
      to Deps if not there already)
    case City
      (find a flight tuple corresponding to a previous flight and add
      to Deps if not there already)
    ;
    case Hotel
      (find a hotel tuple corresponding to the node in a list and add
      to hotel Deps if not there already)
    ;
  endsw
end
```

[0058] In the third step, if the set of dependent flights is null, then go to step 4. Else go to step 5. In the fourth step, departure window is set based on requested date and time. The example program code of the coded algorithm is given below:

```
if (Deps = null) then
  if (requested date (r_i) or requested time (t_i) ! = null) then
    if (requested date (r_i) > Today ) then
      [x_st,x_fi] = (12 am r_i, 11:59 pm r_i + Delta)
    end
    if (requested date (r_i) = Today) then
    [x_st,x_fi] = (Now + mct, Now + mct + Delta)
    end
  end
end
```

[0059] In the fifth step, the widebucket departure window may be set based on requested date and time, if any. The example program code of the coded algorithm is given below:

```
if (requested date (r_i)! = null) then
  (start, finish) = (r_i at 12 am, r_i + Delta at 11:59 pm)
end
```

[0060] Start and finish variables are updated based on arrival windows of dependent flights. Basically, start is set to be maximum start and finish is set to be minimum finish of all arrival windows (where finish>start). The example program code of the coded algorithm is given below:

```
foreach (FlightTuple t in Deps) do
    Let start_t and end_t denote the start and end of widebucket arrival
    window, respectively
  if (start_t > start) then
    start = start_t
  end
  if (end_t < finish AND end_t > start) then
    finish = end_t
  end
end
foreach (HotelTuple t in Deps) do
  if (t.checkout date > start) then
    start =t.checkout date
  end
  if (t.checkout date < finish AND t.checkout date > start) then
    finish = t.checkout date
  end
end
```

[0061] If the requested date (r_i) falls significantly outside the interval (start, finish), the scheduler may build the departure window on r_i. Otherwise, $[x_{st},x_{fi}]$ is equal to (start+mct, finish+mct+Delta).

WideBucket Arrival Window.

[0062] The shortest and longest flying times are determined for the flight F_{i}. The shortest and longest flying time may be fly_{s}, and the longest flying time may be fly_{l}. Thereafter, the time zone difference, cliff, is determined. The time zone difference is an integer number (e.g., number of minutes) depending on regions of departure and arrival airports. Based on the combining, the arrival window may be (x_{st}+fly_{s}+diff, x_{fi}+fly_{l}+diff).

OptimalBucket Window.

[0063] The optimal flight departure window or the optimal flight arrival window (or both) is essentially a narrower time window for when the passenger wants to fly, based on dates and times requested in the itinerary request. Flights found in larger window may be ranked based on whether a flight fits into this narrower window. This process is similar to creation of the flight windows created earlier, with the following difference: if a flight cannot be found in a preferred window, reasonable flights falling outside the optimal window still can be obtain. In conventional methods, these reasonable flights are missed and an empty solution is provided.

[0064] The wide departure window may be determined as [wd_{st}, wd_{fi}], the wide arrival window may be determined as [wa_{st}, wa_{fi}], the narrow departure window may be determined as [nd_{st}, nd_{fi}], and the narrow arrival window may be determined as [na_{st}, na_{fi}].

[0065] The algorithm for determining the optimalbucket windows may include obtaining wider flight windows (departure and arrival):

```
Set [$nd_{st}$,$nd_{fi}$]$=$[$wd_{st}$,$wd_{fi}$]
    initially.
```

[0066] The wider flight windows may be switched based on a type of the itinerary request. Intervals may be handled by going through departure before x1 and departure after x2 separately. Similar steps are performed for arrival intervals:

[0067] (a) Depart on x

[0068] check that $nd_{st}<x<nd_{fi}$

[0069] $(nd_{st}, nd_{fi})=[nd_{st}, x]$

[0070] (b) Depart Before x

[0071] check that $nd_{st}<x<nd_{fi}$

[0072] $(nd_{st}, nd_{fi})=[nd_{st}, x]$

[0073] (c) Depart after x

[0074] check that $nd_{st}<x<nd_{fi}$

[0075] $(nd_{st}, nd_{fi})=[nd_{st}, x]$

[0076] (d) Arrive on x

[0077] check that $na_{st}<x<na_{fi}$

[0078] $(na_{st}, na_{fi})=[na_{st}, x]$

[0079] (e) Arrive Before x

[0080] check that $na_{st}<x<na_{fi}$

[0081] $(na_{st}, na_{fi})=[na_{st}, x]$

[0082] (f) Arrive after x

[0083] check that $na_{st}<x<na_{fi}$

[0084] $(na_{st}, na_{fi})=[x, na_{fi}]$

Partition Algorithm.

[0085] The following program code may be used for initiating the partition algorithm, Cartesian product algorithm, and content search:

```
foreach (Passenger) do
  Partition Algorithm
end
Content Search
  Ranking Partition Subset Solutions
foreach (Passenger) do
  Cartesian Product Algorithm
end
```

[0086] The partition algorithm, Cartesian product algorithm, and ranking are described in more detail below.

[0087] The partition refers to groupings of flight tuples. More specifically, the method **400** can include selecting a partition of the flight tuples for each passenger. Hotel tuples are not part of the partition. There is a possibility to perform more than one partition. Input data for a partition algorithm may include a level matrix. Output data of the partition algorithm is a combination of partitions of flight tuples and/or individual flight tuples. Let Ti is a trip, such as a trip by plane, train, and so forth. Using a depth-first search (DFS) algorithm to find cycles may miss combinations of the form (YYZ-LHR, CDG-YYZ) and (LHR-CDG) corresponding to (T1, T3) and (T2). This may be caused by the fact that DFS follows the path and includes LHR-CDG in the path when creating the cycle. Therefore, the DFS algorithm may produce YYZ-LHR-CDG-YYZ->(T1, T2, T3).

[0088] In example embodiments, other ways of finding potential return combinations of partitions of flight tuples that are not path dependent may be used.

Example Partition Algorithm 1.

[0089] Two flight nodes Fi and Fj may be related as follows: Fi origin city=Fj destination city. This relation may be not an equivalence relation since the relation is not symmetric. However, the relation can partition the main set into meaningful subsets based on potential cycles. The input data for the partition algorithm may include a list of flight nodes. The output data may include a list of subsets of flight nodes. During execution of the partition algorithm, a list of subsets may be defined and called PartitionSubsets. Furthermore, the number of flight nodes may be counted. In an example embodiments, the following program code may be used for this purpose:

```
for (inti = 0; i < Count; i++) do
  for (int j = 0; j < Count; j++) do
  if (i < j) then
    if (Fi.origin city = Fj.destination city) then
      if there is a set in PartitionSubsets containing Fi or Fj
      (but not both)
      then add Fi or Fj to this set
      if no such set then create a new set, add Fi and Fj to
      the new set.
      Add set to PartitionSubsets
      end
    end
  end
end
```

[0090] For any flight nodes not added to a set, single flight sets may be created and added to PartitionSubsets.

[0091] The partition algorithm may produce the same result as creating a graph whose nodes are trips Ti and with edges between two trips if origin city of one trip is equal to destination city of another trip. The new set to be partitioned may consist of disjoint connected components of the graph, which are the PartitionSubsets in the partition algorithm described above.

Example Partition Algorithm 2.

[0092] Execution of the partition algorithm may include finding all pairs of trips where origin of the first trip is the same as destination of the second trip. Furthermore, all disjoint combinations of pairs of trips may be found. The leftover trips not in pairs may be added. Thereby, one or more modified sets may be obtained, where pairs of trips are treated as a single element. Furthermore, all partitions for each modified set may be found. Finally, edge cases may be added.

Cartesian Product Algorithm.

[0093] The Cartesian product algorithm, also referred herein to as a Heuristic algorithm, may combine solutions across the partition subsets and hotel solutions to form a complete flight (and hotel) solution for the passenger. The Cartesian product algorithm may include taking a cross product of solutions across the partition subset solution buckets for flights (A×B×C). Furthermore, each solution obtained may be checked for correctness (e.g., one flight does not depart before a previous dependent flight arrives). These operations are illustrated in FIG. **4**, namely the method **400** may include determining feasible itinerary solutions by the scheduler based on the analysis at operation **460**. The feasible solutions may be stored in a partition solution bucket. The Cartesian product algorithm may further include taking a cross product of full solutions with a list of hotel solutions to combine flights and hotels correctly.

[0094] As shown in FIG. **4**, at operation **470**, the feasible itinerary solutions may be ranked based on passenger preferences. In an example embodiment, the passenger preferences are determined based on the itinerary request or a preexisting selection. More specifically, upon execution of the Cartesian product algorithm, the ranking is applied to feasible solutions provided by Cartesian product algorithm. A predetermined number of the top itinerary solutions may

be selected. If identical dependencies associated with each of the passengers are identified, such as the same flight and hotel dependencies among the passengers, an itinerary wide Cartesian product algorithm may be initiated across the partition solutions and the feasible solutions may be ranked based on the identical dependencies. In some example embodiments, a conventional ranking procedure may be performed. At operation **480**, at least one itinerary solution selected from the feasible itinerary solutions based on the ranking may be presented to the one or more passengers.

Change Management Module.

[0095] Changes to an existing itinerary network can take almost any direction: from changes to the resultant itinerary network to date and time change, adding companions to the itinerary or removing companions of the itineraries.

[0096] Change management of the itinerary network is handled by the change management module using the following steps.

[0097] First, a new itinerary network may be generated by the parser. This new itinerary network may be typically created by making changes to an existing itinerary network.

[0098] Second, the scheduler may generate a new Cartesian product solution by going through the steps of pre-scheduling, resource independent scheduling, partitioning, content searches and Cartesian product, as may be suitable for the new itinerary network. The Cartesian product may be performed on the new itinerary network independently from the existing itinerary network or sets of itinerary solutions associated with the existing itinerary network.

[0099] Third, previously booked flights and hotels, if it is deemed suitable to retain the flights and hotels, may be given additional scores where flights and hotels form part of the new Cartesian product solution for the new itinerary network. In this manner, they the flights and hotels may naturally be a part of a solution space of the new itinerary network. In an example embodiment, inclusion of previously booked flights and hotels may be congruent, in other example embodiments, inclusion of previously booked flights and hotels may be not congruent.

[0100] As a result of selection of a member of the Cartesian product as the itinerary solution, one or more of the following functions may occur: (a) new partitions may be created; (b) some old partitions may be retained; and (c) old partitions may be deleted (i.e., cancelled).

[0101] The overall process performed by the change management module is similar to the process performed by the original module. The operations are performed by the parser and the scheduler (pre-scheduling, resource independent scheduling, and content search). The operation of the change management module is described with reference to one passenger, however, a plurality of passengers also may be analyzed by the change management module.

[0102] The parser may receive an itinerary request from a passenger and identify a preexisting itinerary network associated with the passenger. The preexisting itinerary network may be associated with one or more previous itinerary requests associated with the passenger. The input data for the change management module may include a previous itinerary that may include a partition P_{old} associated with the previous itinerary and a booked solution (denoted as bookedSolutions). The input data may further include an itinerary object associated with a new itinerary request received by the change management module. The process performed by the change management module may include running the resource independent scheduling to create time windows for flights and hotels in a new itinerary network (as in the process performed by the original module). The process may further include running a partition algorithm to create a partition for the new network P_{new}. The process may continue with running a content search to fill itinerary solution buckets for partition subsets of P_{new}. The content search may be unrestricted by the booked old itinerary solution at this stage. The process may further include running a Cartesian product algorithm to obtain feasible itinerary solutions for the new itinerary network. Based on the feasible itinerary solutions, the preexisting itinerary network may be updated for further use during processing of further itinerary requests.

[0103] The following code can be used to perform these operations:

```
foreach (Solution in CartesianProduct) do
  foreach (Flight in solution.Flights) do
    If a flight exists in Booked Solution, then add one to
    ChangeManagementRank field of the solution
    Else Do Nothing
  end
end
```

[0104] In example embodiment, the ranking may be combined with conventional ranking operations. Different types of ranking may have an equal weight or a predetermined weight in a combined ranking. Finally, the solutions may be scored and ranked based on the score.

[0105] FIG. **8** is a block diagram showing various modules of the system **800** for translating user requests into itinerary solutions, in accordance with certain embodiments. The system **800** may include a processor **810**, a parser **820**, a scheduler **830**, and an optional database **840**. The processor **810** may include a programmable processor, such as a microcontroller, central processing unit (CPU), and so forth. In other embodiments, the processor **810** may include an application-specific integrated circuit or programmable logic array, such as a field programmable gate array designed to implement the functions performed by the system **800**.

[0106] The processor **810** may be operable to receive an itinerary request associated with one or more passengers. The parser **820**, being in communication with the processor **810**, may be operable to parse the itinerary request to create an itinerary network associated with the itinerary request. The itinerary network may include at least two or more nodes and dependencies between two or more nodes. The two or more nodes may be selected from a group comprising: an origin city, a destination city, a hotel, a date, a time, an airline, a connection between flights, and so forth.

[0107] In example embodiments, the scheduler **830**, being in communication with the processor **810** and the parser **820**, may be further operable to organize the two or more nodes into levels based on the dependencies between two or more nodes to create a level matrix of the itinerary network. Therefore, the dependencies between two or more nodes may include at least level dependencies. The scheduler **830** may also create a topology of the itinerary network that may include at least an ordered list of the two or more nodes. The topology of the itinerary network may be created based on the level matrix. In further example embodiments, the

scheduler **830** may be operable to classify each of the nodes into a main node and a child node based on the dependencies between the nodes. The child node may be dependent on the main node.

[0108] Thereafter, the scheduler **830** may be operable to process the itinerary network using the topology to create a plurality of tuple, such as flight tuples and hotel tuples. In an example embodiment, the flight tuples may include at least a departure date, a departure time, an arrival date, and an arrival time. The hotel tuples may include at least a check-in date, a check-in time, a check-out date, and a check-out time.

[0109] The scheduler **830** may be further operable to analyze the plurality of tuples based on the itinerary request and the dependencies between two or more nodes. In an example embodiment, the analysis may include separation of the plurality of tuples into groups. Each group may correspond to one of the passengers. Furthermore, a partition of the flight tuples may be selected for each of the one or more passengers.

[0110] Based on the analysis, the scheduler **830** may be operable to determine feasible itinerary solutions. In particular, based on the hotel tuples and the partition of the flight tuples, a content search may be performed to determine the feasible itinerary solutions for each of the one or more passengers. The feasible itinerary solutions may be ranked by the scheduler **830** based on passenger preferences. In an example embodiment, the passenger preferences are determined based on the itinerary request or a preexisting selection. The scheduler **830** may further identify identical dependencies associated with each of the one or more passengers based on the dependencies between two or more nodes. In this embodiment, the ranking may be further based on the identical dependencies.

[0111] The processor **810** may be further operable to present at least one itinerary solution to the one or more passengers. The at least one itinerary solution may be selected from the feasible itinerary solutions based on the ranking.

[0112] In some example embodiments, the parser **820** may be further operable to identify a preexisting itinerary network associated with the passengers. In particular, the preexisting itinerary network may be associated with previous itinerary requests associated with the passengers. Upon identification of the preexisting itinerary network, the parser **820** may update the preexisting itinerary network based on the feasible itinerary solutions selected in response to the itinerary request of the passenger.

[0113] FIG. **9** shows a diagrammatic representation of a computing device for a machine in the exemplary electronic form of a computer system **900**, within which a set of instructions for causing the machine to perform any one or more of the methodologies discussed herein can be executed. In various exemplary embodiments, the machine operates as a standalone device or can be connected (e.g., networked) to other machines. In a networked deployment, the machine can operate in the capacity of a server or a client machine in a server-client network environment, or as a peer machine in a peer-to-peer (or distributed) network environment. The machine can be a PC, a tablet PC, a set-top box, a cellular telephone, a digital camera, a portable music player (e.g., a portable hard drive audio device, such as an Moving Picture Experts Group Audio Layer 3 player), a web appliance, a network router, a switch, a bridge, or any machine capable of executing a set of instructions (sequential or otherwise) that specify actions to be taken by that machine. Further, while only a single machine is illustrated, the term "machine" shall also be taken to include any collection of machines that individually or jointly execute a set (or multiple sets) of instructions to perform any one or more of the methodologies discussed herein.

[0114] The computer system **900** includes a processor or multiple processors **902**, a hard disk drive **904**, a main memory **906**, and a static memory **908**, which communicate with each other via a bus **910**. The computer system **900** may also include a network interface device **912**. The hard disk drive **904** may include a computer-readable medium **920**, which stores one or more sets of instructions **922** embodying or utilized by any one or more of the methodologies or functions described herein. The instructions **922** can also reside, completely or at least partially, within the main memory **906** and/or within the processors **902** during execution thereof by the computer system **900**. The main memory **906** and the processors **902** also constitute machine-readable media.

[0115] While the computer-readable medium **920** is shown in an exemplary embodiment to be a single medium, the term "computer-readable medium" should be taken to include a single medium or multiple media (e.g., a centralized or distributed database, and/or associated caches and servers) that store the one or more sets of instructions. The term "computer-readable medium" shall also be taken to include any medium that is capable of storing, encoding, or carrying a set of instructions for execution by the machine and that causes the machine to perform any one or more of the methodologies of the present application, or that is capable of storing, encoding, or carrying data structures utilized by or associated with such a set of instructions. The term "computer-readable medium" shall accordingly be taken to include, but not be limited to, solid-state memories, optical and magnetic media. Such media can also include, without limitation, hard disks, floppy disks, NAND or NOR flash memory, digital video disks, Random Access Memory (RAM), Read-Only Memory (ROM), and the like.

[0116] The exemplary embodiments described herein can be implemented in an operating environment comprising computer-executable instructions (e.g., software) installed on a computer, in hardware, or in a combination of software and hardware. The computer-executable instructions can be written in a computer programming language or can be embodied in firmware logic. If written in a programming language conforming to a recognized standard, such instructions can be executed on a variety of hardware platforms and for interfaces to a variety of operating systems.

[0117] In some embodiments, the computer system **900** may be implemented as a cloud-based computing environment, such as a virtual machine operating within a computing cloud. In other embodiments, the computer system **900** may itself include a cloud-based computing environment, where the functionalities of the computer system **900** are executed in a distributed fashion. Thus, the computer system **900**, when configured as a computing cloud, may include pluralities of computing devices in various forms, as will be described in greater detail below.

[0118] In general, a cloud-based computing environment is a resource that typically combines the computational power of a large grouping of processors (such as within web servers) and/or that combines the storage capacity of a large grouping of computer memories or storage devices. Systems

that provide cloud-based resources may be utilized exclusively by their owners, or such systems may be accessible to outside users who deploy applications within the computing infrastructure to obtain the benefit of large computational or storage resources.

[0119] The cloud may be formed, for example, by a network of web servers that comprise a plurality of computing devices, such as a client device, with each server (or at least a plurality thereof) providing processor and/or storage resources. These servers may manage workloads provided by multiple users (e.g., cloud resource consumers or other users). Typically, each user places workload demands upon the cloud that vary in real-time, sometimes dramatically. The nature and extent of these variations typically depends on the type of business associated with the user.

[0120] It is noteworthy that any hardware platform suitable for performing the processing described herein is suitable for use with the technology. The terms "computer-readable storage medium" and "computer-readable storage media" as used herein refer to any medium or media that participate in providing instructions to a CPU for execution. Such media can take many forms, including, but not limited to, non-volatile media, volatile media and transmission media. Non-volatile media include, for example, optical or magnetic disks, such as a fixed disk. Volatile media include dynamic memory, such as system RAM. Transmission media include coaxial cables, copper wire, and fiber optics, among others, including the wires that comprise one embodiment of a bus. Transmission media can also take the form of acoustic or light waves, such as those generated during radio frequency (RF) and infrared (IR) data communications. Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, a hard disk, magnetic tape, any other magnetic medium, a CD-ROM disk, digital video disk, any other optical medium, any other physical medium with patterns of marks or holes, a RAM, a Programmable Read-Only Memory, an Erasable Programmable Read-Only Memory (EPROM), an Electrically Erasable Programmable Read-Only Memory, a FlashEPROM, any other memory chip or data exchange adapter, a carrier wave, or any other medium from which a computer can read.

[0121] Various forms of computer-readable media may be involved in carrying one or more sequences of one or more instructions to a CPU for execution. A bus carries the data to system RAM, from which a CPU retrieves and executes the instructions. The instructions received by system RAM can optionally be stored on a fixed disk either before or after execution by a CPU.

[0122] Computer program code for carrying out operations for aspects of the present technology may be written in any combination of one or more programming languages, including an object oriented programming language such as Java, Smalltalk, C++ or the like and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a LAN or a WAN, or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider).

[0123] The corresponding structures, materials, acts, and equivalents of all means or steps plus function elements in the claims below are intended to include any structure, material, or act for performing the function in combination with other claimed elements as specifically claimed. The description of the present technology has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the disclosure. Many modifications and variations will be apparent to those of ordinary skill in the art without departing from the scope and spirit of the disclosure. Exemplary embodiments were chosen and described in order to best explain the principles of the present technology and its practical application, and to enable others of ordinary skill in the art to understand the disclosure for various embodiments with various modifications as are suited to the particular use contemplated.

[0124] Aspects of the present technology are described above with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems), and computer program products according to embodiments of the disclosure. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0125] These computer program instructions may also be stored in a computer readable medium that can direct a computer, other programmable data processing apparatus, or other devices to function in a particular manner, such that the instructions stored in the computer readable medium produce an article of manufacture including instructions which implement the function/act specified in the flowchart and/or block diagram block or blocks.

[0126] Thus, computer-implemented methods and systems for translating user requests into itinerary solutions are described. Although embodiments have been described with reference to specific exemplary embodiments, it will be evident that various modifications and changes can be made to these exemplary embodiments without departing from the broader spirit and scope of the present application. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. A system for changing an itinerary based on a user itinerary change request, the system comprising:

a processor configured to:

receive an itinerary network associated with one or more passengers; and

receive the user itinerary change request associated with the itinerary network;

a parser configured to:

generate an itinerary object associated with the user itinerary change request; and

modify the itinerary network based on the itinerary object; and

a scheduler configured to:

process the modified itinerary network using a topology of the itinerary network to create a plurality of tuples, the plurality of tuples including at least flight tuples and hotel tuples;

perform a content search for the plurality of tuples for each of the one or more passengers; and

generate feasible itinerary solutions based on results of the content searches.

2. The system of claim **1**, wherein the flight tuples include at least a departure data, a departure time, an arrival date, and an arrival time; and wherein the hotel tuples include at least a check-in date, a check-in time, a check-out date, and a check-out time.

3. The system of claim **1**, wherein the itinerary network comprises two or more nodes and dependencies between the two or more nodes, the two or more nodes being selected from a group of: an origin city, a destination city, a hotel, a date, a time, an airline, and a connection between flights.

4. The system of claim **3**, wherein the topology of the itinerary network comprises at least an ordered list of the two or more nodes.

5. The system of claim **3**, wherein the scheduler is further configured to create an adjacency matrix of the modified itinerary network based on a classification of the two or more nodes based on the dependencies between the two or more nodes.

6. The system of claim **3**, wherein the scheduler is further configured to create a level matrix of the modified itinerary network by selecting main nodes and child nodes that depend on the main nodes from the two or more nodes in the modified itinerary network and wherein the topology is based on the level matrix.

7. The system of claim **1**, wherein the user itinerary change request is provided via at least one of: a natural language, a typed text, and a selection of preexisting options.

8. The system of claim **1**, wherein the user itinerary change request is further associated with a group of passengers.

9. The system of claim **1**, wherein the scheduler is further configured to:

partition, for each of the one or more passengers, the flight tuples into one or more disjoint partition subsets, the disjoint partition subsets determined based on a property of interest; and

perform the content search for each disjoint partition subset of the flight tuples and for each of the hotel tuples.

10. The system of claim **9**, wherein the scheduler is further configured to perform the content search across all disjoint partition subsets for all passengers.

11. A computer-implemented method for changing an itinerary based on a user change request, the method comprising:

receiving, by a processor, an itinerary network associated with one or more passengers;

receiving, by the processor, a user itinerary change request associated with the itinerary network;

generating, by a parser, an itinerary object associated with the user itinerary change request;

modifying, by the parser, the itinerary network based on the itinerary object;

processing, by a scheduler, the itinerary network using a topology of the itinerary network to create a plurality of tuples, the plurality of tuples including at least flight tuples and hotel tuples;

performing, by the scheduler, a content search for the plurality of tuples for each of the one or more passengers; and

generating, by the scheduler, feasible itinerary solutions based on results of the content searches.

12. The method of claim **11**, wherein the flight tuples include at least a departure data, a departure time, an arrival date, and an arrival time; and

wherein the hotel tuples include at least a check-in date, a check-in time, a check-out date, and a check-out time.

13. The method of claim **11**, wherein the itinerary network comprises at least two or more nodes and dependencies between the at least two or more nodes, the at least two or more nodes being selected from a group of: an origin city, a destination city, a hotel, a date, a time, an airline, and a connection between flights.

14. The method of claim **13**, wherein the topology of the itinerary network comprises at least an ordered list of the two or more nodes.

15. The method of claim **13**, wherein comprising creating, by the scheduler, an adjacency matrix of the modified itinerary network based on a classification of the two or more nodes based on the dependencies between the two or more nodes.

16. The method of claim **13**, further comprising creating, by the scheduler, a level matrix of the modified itinerary network by selecting main nodes and child nodes that depend on the main nodes from the two or more nodes in the modified itinerary network and wherein the topology is based on the level matrix.

17. The method of claim **11**, wherein the user itinerary change request is provided via at least one of: a natural language, a typed text, and a selection of preexisting options.

18. The method of claim **11**, wherein the user itinerary change request is further associated with a group of passengers.

19. The method of claim **11**, further comprising performing, by the scheduler, a content search across all partition subsets for all passengers.

20. The method of claim **11**, further comprising:

partitioning, by the scheduler, for each of the one or more passengers, the flight tuples into one or more disjoint partition subsets, the disjoint partition subsets determined based on a property of interest; and

performing, by the scheduler, the content search for each disjoint partition subset of the flight tuples and for each of the hotel tuples.

* * * * *