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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 July 2018 (26.07.2018)

WIPO | PCT

(10) International Publication Number
WO 2018/136371 A1

(51) International Patent Classification:
*H03M 7/46* (2006.01)

(21) International Application Number:
PCT/US2018/013750

(22) International Filing Date:
16 January 2018 (16.01.2018)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
201710044988.5 20 January 2017 (20.01.2017) CN

(71) Applicant: **MICROSOFT TECHNOLOGY LICENSING, LLC** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(72) Inventors: **ZHOU, Hucheng**; MICROSOFT TECHNOLOGY LICENSING, LLC, One Microsoft Way, Redmond, Washington 98052-6399 (US). **YE, Ting**; MICROSOFT TECHNOLOGY LICENSING, LLC, One Microsoft Way, Redmond, Washington 98052-6399 (US).

(74) Agent: **MINHAS, Sandip S.** et al.; MICROSOFT TECHNOLOGY LICENSING, LLC, One Microsoft Way, Redmond, Washington 98052-6399 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

(54) Title: COMPRESSED ENCODING FOR BIT SEQUENCE

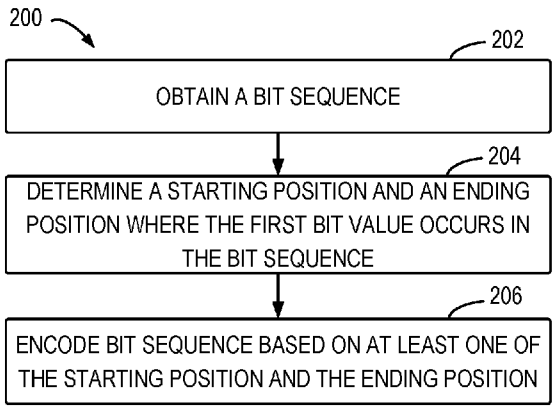


FIG. 2

(57) Abstract: In implementations of the subject matter described herein, a method and device for compressed encoding for a bit sequence are proposed. After the bit sequence is obtained, the starting position and ending position where a particular bit value occurs in the bit sequence are determined, and then the bit sequence is encoded based on the starting position and ending position. In accordance with implementations of the subject matter described herein, effective compression for the bit sequence can be achieved by determining the starting position and ending position where a particular bit value occurs in the bit sequence, thereby reducing the length of the bit sequence. Therefore, implementations of the subject matter described herein can not only reduce computational complexity, but also improve parallel data processing capability.

WO 2018/136371 A1

*[Continued on next page]*

**Declarations under Rule 4.17:**

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

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

**Published:**

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

# COMPRESSED ENCODING FOR BIT SEQUENCE

## BACKGROUND

**[0001]** A bit is the smallest data storage unit in a computer, which represents a binary digit and is valued as 0 or 1. A byte is a basic unit for data storage in the computer and usually taken as a unit for storing and interpreting information in the computer. A byte (B) equals to eight bits (b) and may generally store one English character or a half of one Chinese character. Generally, a large amount of information needs to be stored, transmitted and processed in the computer, and thus kilobyte (KB), megabyte (MB), gigabyte (GB) and terabyte (TB) are usually used as the data storage units.

**[0002]** A decision tree is a tree-structure for handling categorizing, regressing and sorting problems, which is a binary search tree consist of the branch nodes, the left and right child nodes. The branch is traversed from a branch node to another branch node or a leaf node, and the selection of the branch is based on the decision made at the branch node. The example decision comprises comparing two values, such as a feature value and an attribute value (also referred to as a threshold). If the feature value is smaller than or equal to the threshold, then the left child node is selected; if the feature value is larger than the threshold, then the right child node is selected until the leaf node is traversed. The leaf node represents an output or endpoint of the decision tree, and the example output is an output value or a score for the decision tree.

**[0003]** A boosting tree ensemble model is a model of a boosting method which uses a decision tree as a basic classifier, and it includes a plurality of decision trees and makes a decision collectively by aggregating the results of a plurality of decision trees. The boosting tree ensemble may be used to sort documents during the document searching, and it may be applied to other technical fields such as gesture recognition, voice recognition, data mining and the like.

## SUMMARY

**[0004]** It is noted that the decision tree model (such as a boosting tree ensemble model) is widely used for web-based applications. With the constant increase of web contents, the depth and scale of each tree in the decision tree model grow accordingly, which demands a large quantity of computational resources and storage resources to run the decision tree model. Different from the traditional approach which only uses the bit sequences to represent feature vectors for the nodes in the tree, the subject matter described herein performs compressed encoding to bit sequences of feature vectors for nodes. By means of

compressed encoding based on the starting position and the ending position associated with a particular bit value, the processing speed and storage space of the boosting tree ensemble model can be optimized, which significantly differs from any currently known scheme in terms of working principle and mechanism.

**[0005]** In implementations of the subject matter described herein, there is proposed a method and device for compressed encoding for a bit sequence. After the bit sequence is obtained, the starting position and ending position where a particular bit value occurs in the bit sequence are determined, and then the bit sequence is encoded on this basis. In accordance with implementations of the subject matter described herein, effective compression for the bit sequence can be achieved by determining the starting position and ending position where a particular bit value occurs in the bit sequence, thereby reducing the length of the bit sequence. Therefore, implementations of the subject matter described herein can not only reduce computational complexity but also improve parallel data processing capability.

**[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 to limit the scope of the claimed subject matter.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0007]** With reference to the drawings and the detailed description below, the above and other features, advantages and aspects of implementations of the subject matter described herein will become more apparent. In the drawings:

**[0008]** Fig. 1 shows a block diagram of a computing system/server capable of implementing one or more implementations of the subject matter described herein;

**[0009]** Fig. 2 shows a flowchart of a method for encoding a bit sequence according to implementations of the subject matter described herein;

**[0010]** Fig. 3 shows an example diagram of a decision tree according to implementations of the subject matter described herein;

**[0011]** Fig. 4 shows a flowchart of a method for encoding a bit sequence according to another implementation of the subject matter described herein;

**[0012]** Fig. 5 shows an example diagram for a data structure configuration according to implementations of the subject matter described herein;

**[0013]** Fig. 6 shows a flowchart of a method for determining a plurality of outputs for a plurality of inputs in parallel according to implementations of the subject matter described

herein;

**[0014]** Fig. 7 shows a schematic diagram for determining a plurality of outputs for a plurality of inputs in parallel according to implementations of the subject matter described herein; and

**[0015]** Fig. 8 shows a schematic block diagram for a search system for search ranking according to implementations of the subject matter described herein.

**[0016]** Throughout the drawings, the same or similar reference signs are always used to indicate the same or similar elements.

## DETAILED DESCRIPTION

**[0017]** The implementations of the subject matter described herein will be described in details with reference to the drawings. Although the drawings illustrate some implementations of the subject matter described herein, it is to be understood that the subject matter described herein described herein may be implemented in various manners other than the ones described herein. To the contrary, these implementations are provided for a more thorough and comprehensive understanding of the subject matter described herein. It should be understood that the drawings and implementations of the subject matter described herein are only for the purpose of illustration, rather than suggesting any limitations on the scope of the subject matter.

**[0018]** As used herein, the term "includes" and its variants are to be read as open-ended terms that mean "includes, but is not limited to." The term "based on" is to be read as "based at least in part on." The term "one example implementation" is to be read as "at least one example implementation." The term "another implementation" is to be read as "at least one other implementation." The term "some implementations" is to be read as "at least some implementations". Relevant definitions of other terms will be given in the following depiction.

**[0019]** Conventionally, a decision tree model needs to traverse each tree of the decision tree model to generate the output for the received input. At each branch node, the feature value in the input and the threshold of the branch node are compared to determine the next node. However, since the operation at the child node is performed after the operation at its parent node, the child node has controlling dependency relation on its parent node. Furthermore, since a random determining result occurs at each branch node, the cache cannot be utilized efficiently.

**[0020]** An improvement made to the decision tree model is a node representation method based on the bit sequence, in which a feature vector of each node is represented with a bit

sequence. In this manner, the controlling dependency relation between the child node and the parent node can be ignored, and the output is determined only based on the bit sequence of a certain type of nodes (for example, false nodes). However, in this method, the length of bit sequence of each node must be equal to the total number of leaf nodes in the tree. As such, if the scale of the tree is large, then the length of each bit sequence is also very large such that a large number of computational resources are required to process these bit sequences and a large quantity of storage resources are needed to store these bit sequences. Moreover, if the length of bit sequence is larger than the maximum length that can be processed by the processor of the computer at a time, it would be impossible to perform parallel processing operation. Thus, when the scale of the decision tree model is very large, the traditional tree traversing method and the traditional node representation method based on bit sequence have the problems of long operation time and low processing efficiency.

**[0021]** For this purpose, the subject matter described herein provides a method and device for encoding a bit sequence. After the bit sequence is obtained, the starting position and ending position where a particular bit value occurs in the bit sequence are determined, and then the bit sequence is encoded based on the starting position and ending position. In accordance with implementations of the subject matter described herein, effective compression for a bit sequence can be achieved by determining the starting position and ending position where a particular bit value occurs in the bit sequence, thereby reducing the length of the bit sequence. Therefore, implementations of the subject matter described herein can not only reduce computational complexity but also improve parallel data processing capability.

**[0022]** Moreover, according to implementations of the subject matter described herein, the byte is used as a unit for encoding the bit sequence so as to be suitable for internal processing of the computer. Meanwhile, when the starting byte position and the ending byte position are the same, only the starting byte and its position are used to encode the bit sequence, which can further compress the bit sequence. Furthermore, according to implementations of the subject matter described herein, a plurality of inputs for a single node may be processed at the same time and the cache can be utilized efficiently after the vector transposition, thereby achieving parallel processing of the plurality of inputs in order to generate a plurality of outputs in parallel. Therefore, implementations of the subject matter described herein enable parallel processing during the whole process in a decision tree model.

**[0023]** The basic principles and several example implementations of the subject matter

described herein are described in the following text with reference to Figs. 1-8. Fig. 1 shows a block diagram of a computing system/server 100 capable of implementing one or more implementations of the subject matter described herein. It is to be understood that the computing system/server 100 shown in Fig. 1 is only an example, not intended to limit the function and scope of the implementations described herein in any manner.

**[0024]** A shown in Fig. 1, the computing system/server 100 is a general form of computing device. The components of the computing system/server 100 may include, but are not limited to, one or more processors or processing units 110, a memory 120, storage 130, one or more communication units 140, one or more input devices 150, and one or more output devices 160. The processing unit 110 may be an actual or a virtual processer and can perform various processing according to programs stored in the memory 120. In a multi-processor system, a plurality of processing units execute machine-executable instructions in parallel to improve the parallel processing capability of the computing system/server 100.

**[0025]** The computing system/server 100 typically includes a variety of machine readable mediums. Such mediums may be any available medium that is accessible by the computing system/ server 100, including volatile and non-volatile mediums, removable and non-removable mediums. The memory 120 may be volatile memory (such as a register, a cache, a random-access memory (RAM)), a non-volatile memory (such as a read only memory (ROM), an electrically erasable programmable read only memory (EEPROM), a flash memory), or some combination thereof. The storage 130 may be removable or non-removable, and may include machine readable medium such as flash drives, magnetic disks or any other medium which may be used to store information and/or data (such as model 170) and which may be accessed within the computing system/server 100.

**[0026]** The computing system/server 100 may further include other removable/non-removable, volatile/non-volatile storage mediums. Although not shown in Fig. 1, a magnetic disk driver for reading from or writing to a removable, non-volatile disk (such as a “floppy disk”), and an optical disk driver for reading from or writing to a removable, non-volatile optical disk may be provided. In these cases, each driver may be connected to the bus via one or more data medium interfaces (not shown). The memory 120 may include one or more program products 125 having a set of one or more program modules that are configured to carry out the methods or functions of various implementations of the subject matter described herein.

**[0027]** The communication unit 140 communicates with a further computing device via

the communication media. Additionally, functions of components of the computing system/server 100 may be implemented by a single computing cluster or multiple computing machines that are connected communicatively for communication. Therefore, the computing system/server 100 may be operated in a networking environment using a logical link with one or more other servers, personal computers (PCs) or another general network node.

**[0028]** The input device 150 may include one or more input devices, such as a mouse, a keyboard, a tracking ball and the like. The output device 160 may be one or more output devices, such as a display, a loudspeaker, a printer, and the like. As required, the computing system/server 100 may also communicate, via the communication unit 140, with one or more external devices (not shown, such as a storage device, a display device and the like), one or more devices that enable users to interact with the computing system/server 100, or any devices that enable the computing device 100 to communicate with one or more other computing devices (for example, a network card, a modem, and the like). Such communication is performed via an input/output (I/O) interface (not shown).

**[0029]** As shown in Fig. 1, the storage 130 is stored with a model 170 which contains a decision tree model(s) 175, such as a boosting tree ensemble model. The computing system/server 100 may receive a input(s) 180 via the input device 150, for instance, a plurality of inputs $\{i_0, i_1, i_2, \ldots\}$, each of which includes a plurality of features $\{f_0, f_1, f_2, \ldots\}$. As shown in Fig. 1, the value of feature $f_0$ of $i_0$ may be 3.8, the value of feature $f_1$ of $i_0$ may be 26.4 and the value of feature $f_2$ of $i_0$ may be 2.0. Next, the computing system/server 100 uses the decision tree model 175 to process the input 180 and generates a plurality of outputs 190 via the output device 160, such as output value $\{o_0, o_1, o_2, \ldots\}$, where each output may correspond to one input. For example, $o_0$ (such as 2.0) is an output value of the input $i_0$. In the following description, example implementations for generating the output(s) 190 based on the decision tree model 175 is described in detail with reference to Figs. 2-8.

**[0030]** Fig. 2 shows a flowchart of a method 200 for encoding a bit sequence according to implementations of the subject matter described herein. It is to be understood that method 200 may be performed by processing unit 110 described with reference to Fig. 1.

**[0031]** At 202, a bit sequence is obtained. For example, in the decision tree model based on the node representation of the bit sequence, each node has a bit sequence representing its feature vector, and one or more bit sequences of one or more nodes may be obtained from the decision tree model. The bit sequence consists of a plurality of bits,

including a first bit value (such as 0) and a second bit value (such as 1). For example, an example bit sequence $b_e$ may be 11111111110000000111111111111111, including seven first bit values (namely, 0) and 25 second bit values (namely, 1).

**[0032]** At 204, the starting position and ending position of the first bit value in the bit sequence are determined. In some implementations, the starting position and the ending position of the first bit value per se may be directly determined. For instance, in the example bit sequence $b_e$, the starting position (such as an index) where the first bit value per se appears is 10 while the ending position is 16. In some other implementations, the starting position and the ending position may represent the positions of a range of bits where the first bit value is in. For example, the starting position may be a position of the starting byte including the beginning first bit value (such as 0), and the ending position may be a position of the ending byte including the ending first bit value. Therefore, in the example bit sequence $b_e$, the position of the starting byte where the first bit value appears is 1, while the position of the ending byte where the first bit value appears is 2. It is to be understood that although in the above examples, the position or index for the first bit or byte is defined as 0, those skilled in the art should understand that it may also be defined as 1. The scope of protection of the subject matter described herein is not limited in the aspect of index defining rules. In the following description, an example implementation for determining the starting position and ending position is described in detail with reference to Fig. 4.

**[0033]** At 206, the bit sequence is encoded based on at least one of the starting position and the ending position. In some implementations, under the condition that all the first bit values in the bit sequence exist continuously, the bit sequence $b_e$ may be encoded as [10, 16], which represents that all bits between position 10-16 are the first bit (such as 0) and all bits in the other positions are the second bit (such as 1). In some other implementations, the bit sequence $b_e$ may be encoded as retaining all the bits between the starting position and the ending position. For instance, bit sequence $b_e$ may be also encoded as [10, 16, 0000000]. Alternatively, when the starting position represents the position of the starting byte and the ending position represents the position of the ending byte, the bit sequence $b_e$ may be encoded as associated with at least one of the starting byte position and the ending byte position. For example, the bit sequence $b_e$ may be encoded based on the starting byte and the position of the starting byte.

**[0034]** It is to be understood by those skilled in the art although in method 200, the

bit sequence represents a feature vector in the decision tree, the bit sequence may represent other data with any meaning. Therefore, by determining the starting position and ending position of a particular value in the bit sequence, effective compression for the bit sequence can be achieved, thereby reducing the length of the bit sequence. Therefore, implementations of the subject matter described herein can efficiently reduce both the computational complexity and the storage space.

**[0035]** Fig. 3 shows an example diagram of a decision tree 300 according to implementations of the subject matter described herein. For instance, the decision tree model 175 described with reference to Fig. 1 may include $h$ decision trees $\{T_0, T_1, \ldots T_h\}$, and each decision tree is represented as $T_h=(N_h, L_h)$, where $N_h=\{n_0, n_1, \ldots\}$ represents $|L_h|$-1 branch nodes in the decision tree $T_h$, and $L_h=\{l_0, l_1, \ldots\}$ represents $|L_h|$ leaf nodes in the decision tree $T_h$. The decision tree 300 may be an example decision tree in the decision tree model 175. As shown in Fig. 3, the decision tree 300 includes branch nodes 310, 311, 312, 313, 314, 315, where branch node 310 is a root node, and decision tree 300 further includes leaf nodes 320, 321, 322, 323, 324, 325, 326. It is to be understood that, for the sake of clear illustration, the decision tree 300 only includes six branch nodes and seven leaf nodes, however, the decision tree 300 may contain any number of branch nodes and leaf nodes, for instance, 255 branch nodes and 256 leaf nodes.

**[0036]** Generally, each branch node in the decision tree has a feature identification (for instance, $\{f_0, f_1, f_2, \ldots\}$) and a corresponding attribute value which may be used to determine the Boolean value of the node, also referred to as the threshold (such as $\{\theta_0, \theta_1, \theta_2, \ldots\}$). Moreover, each branch node has a bit sequence representing its feature vector that can be used to determine the position of a leaf node for the input. Each leaf node has an output value (for example $\{e_0, e_1, e_2, \ldots\}$). For example, the branch node 310 has a feature identification $f_0$ and the corresponding threshold $\theta_0$. For a new input $i_0$, if the value of its feature $f_0$ is smaller than the threshold $\theta_0$, then the branch node 310 is regarded as a true node; otherwise, the branch node 310 is regarded as a false node. In the traditional tree traversing method, if a certain branch node is determined as a true node, then the left child node is selected; otherwise, the right child node is selected. In this way, the process traverses from the root node through to the corresponding leaf node.

**[0037]** In the node representation method based on the bit sequence, it is unnecessary to consider the controlling dependency relation between the child node

and the parent node, and the Boolean value of each branch node in a plurality of trees may be determined sequentially, randomly or in parallel. For example, in the example decision tree 300 shown in Fig. 3, by a comparison between the feature value and the threshold, branch nodes 310, 311 and 315 are determined as false nodes and other nodes are determined as true nodes. In the node representation method based on the bit sequence, logic AND algorithm are performed on the bit sequences for all the false nodes to generate a result for the input. For example, an input causes a result for a tree, and then the position of the leaf node may be determined based on the result in order to generate an output value or a score.

**[0038]** It is to be understood that the bit sequence for each node in the decision tree may be generated by machine learning for the training data. Any machine learning method currently known or to be developed in the future may be used to generate bit sequence of each node. For example, all the default bits in the bit sequence of the decision result for each tree may be set as 1, and the bit value in the bit sequence may be updated with input samples, so that the position of the leaf node may be determined based on the position of the first value "1" in the bit sequence of this leaf node.

**[0039]** Since the length of the feature vector equals to the number of all the leaf nodes in the decision tree, in the case that the scale of the decision tree is large, the bit sequence would also be long. According to implementations of the subject matter described herein, the compressed encoding is performed on the bit sequence so that the length of the feature vector does not necessarily to be equal to the number of leaf nodes in the tree.

**[0040]** Fig. 4 shows a flowchart of a method 400 for encoding a bit sequence according to another implementation of the subject matter described herein. In the method 400, actions 402-408 may be sub-actions of action 204 described with reference to Fig. 2 and actions 412-414 may be sub-actions of action 206 described with reference to Fig. 2. It is to be understood that the method 400 may be performed by the processing unit 110 described with reference to Fig. 1.

**[0041]** At 402, the starting byte including the first bit value is determined. For instance, the above bit sequence $b_e$ may be divided into a plurality of bytes {11111111,11000000,01111111,11111111}, and then the first byte {11000000} including the first bit value is determined as the starting byte. At 404, the index of the starting byte is determined as the starting position. For example, the index of the starting byte {11000000} may be determined as 1.

**[0042]** At 406, the ending byte including the first bit value is determined. For instance, in the bit sequence $b_e$\{11111111,11000000,01111111,11111111\}, the last byte \{01111111\} including the first bit value is determined as the ending byte. At 408, the index of the ending byte is determined as ending position. For instance, the index of the ending byte \{01111111\} may be determined as 2.

**[0043]** At 410, it is determined whether the starting position is the same as the ending position. If they are the same, at 412 the bit sequence is encoded based on the starting byte and the starting position, which indicates that all the first bit values are in a single byte, and it is unnecessary to encode the bit sequence with the ending byte and the ending position. If they are different, then at 414 the bit sequence is encoded based on the starting byte, the starting position, ending byte and ending position. Therefore, according to the method 400 of the subject matter described herein, when the starting byte position and the ending byte position are the same, the bit sequence is encoded only with the starting byte and its position, by which the bit sequence can be further compressed.

**[0044]** In an implementation, for instance, the feature vector of the node may be a bit sequence $b_f$ (11...1100...0011...11) with 256 bits, where the bit sequence consists of three parts, for example, eighty-one continuous 1, seventeen continuous 0, and one hundred and fifty-eight continuous 1, respectively. According to the method 400 described herein, it can be determined that the starting byte including the first bit value 0 is 10000000 with the starting position 10, and the ending byte 00111111 with the ending position 12. Hence, according to implementations of the subject matter described herein, only the starting byte, the ending byte, the starting position and the ending position are used to encode the bit sequence. That is, only $2+2*\lceil (\log_2(\frac{L}{8})/8)\rceil$ bytes are needed to represent the encoded bit sequence, where $L$ represents the total sum of leaf nodes in the decision tree, $\lceil\log_2(\frac{L}{8})/8\rceil$ represents the byte(s) occupied by the starting position or the ending position, and $\lceil\rceil$ represents a sign of rounding up to an integer. In this example, the value of $L$ equals the length of bit sequence, that is, 256. Therefore, the encoded bit sequence merely has a length of about four bits, thereby effectively compressing the original bit sequence.

**[0045]** Generally, Single Instruction Multiple Data (SIMD) has a length of a plurality of bits to process a plurality of bit data at a time. For instance, Streaming SIMD extension (SSE) register supports instruction with 128 bits and Advanced Vector

Extension (AVX) register supports instruction with 256 bits. In the traditional approach, since the length of the bit sequence is the same as the sum of leaf nodes, when the length of a bit sequence reaches a certain value (for example, exceeding 64 bits in the SSE register, or exceeding 128 bits in the AVX register), it is impossible to perform the parallel processing. However, according to implementations of the subject matter described herein, by means of compressed encoding of the bit sequence, the length of the encoded bit sequence can be relatively shortened, thereby processing more bit sequences in parallel. According to implementations of the subject matter described herein, by means of compressed encoding and transposition storage of the bit sequence, the SSE register may, for instance, process 16 bit sequences in parallel and AVX register may process 32 bit sequences in parallel, without being affected by the length of the original bit sequence. Furthermore, in the register that supports 512 bits (for example, AVX-512), the subject matter described herein can at most process 64 bit sequences in parallel.

**[0046]** Fig. 5 shows an example diagram for a data structure configuration 500 according to implementations of the subject matter described herein. For a plurality of trees in the decision tree model, a plurality of nodes with the same feature identification may be combined and sorted by threshold size in an ascending order. Since the numbers of different feature identifications are different, offset indexes should be used to indicate the starting position of each feature identification. As shown in Fig. 5, the offset indexes 510 are used to record the positions of the starting thresholds in thresholds 520 which correspond to feature identifications. For example, with the first two indexes in the offset indexes, the threshold range of feature identification $f_0$ may be determined. As shown in Fig. 5, each threshold corresponds to the tree identifications (IDs) 530 of the respective node and the encoded bit sequences 540 (for example, encoded bit sequences according to the method 200 or 400 described herein). For example, optionally, the length of each encoded bit sequence may be configured as $2+2*\lceil(\log_2(\frac{L}{8}))/8\rceil$, where $L$ represents the sum of leaf nodes in the tree. Alternatively, in the case that the position of the starting byte and the position of the ending byte are the same, the length of each encoded bit sequence may be configured as $1+\lceil(\log_2(\frac{L}{8}))/8\rceil$.

**[0047]** After comparing the feature value and the threshold value, all the false nodes in each tree can be determined, and logic AND operation may be performed on the encoded bit sequences of all the false nodes to generate a resulting sequence(s). As

shown in Fig. 5, for the received input, *h* results 550 may be generated for *h* trees, and the position of the corresponding leaf node may be determined according to each result so as to generate an output value. As indicated by Fig. 5, each leaf node of each decision tree has its respective leaf node output value 560.

**[0048]** According to implementations of the subject matter described herein, the input feature value and the corresponding threshold are compared, and the tree IDs and bit sequences of all the false nodes are recorded. Then, logic AND operation are performed on the bit sequences of all the false nodes in each tree to obtain a result corresponding to each input. After that, the position of the leaf node is determined based on the result, and an output value or score may be generated.

**[0049]** There are a large number of decision trees in the decision tree model 175 and each decision tree includes a plurality of branch nodes, and a great quantity of repetitive nodes are found to be present. That is, some nodes have the same feature identification and the same threshold. In some implementations, a plurality of nodes with the same feature identification and the same threshold may be determined in the decision tree, and then the Boolean value determination operations for a plurality of nodes may be combined. In other words, for a plurality of nodes with exactly the same feature identification and the same threshold, it is only necessary to perform one comparison operation between the feature value and the threshold for these nodes without the need to perform comparison operation for each of the nodes, thereby improving the processing speed of the decision tree. It is to be understood that if the decision tree model is large and there are a lot of the repetitive nodes, then more obvious performance improvement will be brought by the de-repeating operation of the nodes. Besides, since the bit sequence for each node is not necessarily the same, only comparison operation between the feature value and the threshold is combined for a plurality of nodes with the same feature identification and the same threshold, but the logic AND operations for determining the result are not combined.

**[0050]** In some implementations, with reference to the thresholds 520 shown in Fig. 5, after the plurality of thresholds of a plurality of nodes with the same identifier are sorted in an ascending order, if a feature value associated with the feature identification is smaller than a certain threshold, then the comparison between the feature value and thresholds larger than the certain threshold among a plurality of thresholds may be terminated. This is because the comparison operation only needs to find false nodes in the decision tree, and if the feature value is smaller than the certain threshold, then

the feature value is inevitably smaller than other thresholds that are larger than the certain threshold. Therefore, by sorting the plurality of thresholds according to feature identification in an ascending order and only performing a part of comparison operation, the processing speed of the decision tree can be improved.

**[0051]** In some implementations, the data access speed can be improved by prefetching data to the cache. In some implementations, the efficiency of storage hierarchy can be improved by use of blocking. For instance, all the decision trees in the decision tree model 175 may be divided into several blocks so that each block can be placed in the cache on the third level (L3) of the processing unit.

**[0052]** According to implementations of the subject matter described herein, by use of the node representation method based on the bit sequence, the length of the bit sequence can be reduced, which provides condition for vector-based parallel processing. For example, in SSE register, logic AND operations for a plurality of bit sequences can be processed in parallel.

**[0053]** In some implementations, the decision tree may simultaneously receive a plurality of inputs (for example, inputs $\{i_0, i_1, i_2, \ldots\}$), and the plurality of inputs are then processed in parallel for one node to improve parallel processing efficiency. For instance, the threshold of one node may be placed in one SSE register, and then a plurality of received corresponding inputs are placed in another SSE register. To process a plurality of inputs sequentially, the plurality of inputs can be aggregated based on feature identifications such that a plurality of feature values with the same feature identifications are stored continuously. Then, for each input, it can be determined in parallel whether the node is true node or false node. In some implementations, an intermediate vector may be used to store information of bit sequences of false nodes.

**[0054]** In some implementations, after all the false nodes in a decision tree are performed with logic AND operation and a plurality of results for a plurality of inputs are generated, the vector representing a plurality of results may be transposed such that a plurality of bytes at the same position in a plurality results may be stored continuously in the cache. For example, the following Table 1 shows the vector representing a plurality of results $\{v_0, v_1, v_2, \ldots\}$. It can be seen that a plurality of bytes of each result $\{v_0^0, v_0^1, v_0^2, \ldots\}$ are stored continuously. The following Table 2 shows the transposed vector, where a plurality of bytes of each result are not stored

continuously. Instead, in Table 2, a plurality of bytes at the same position in a plurality of results are stored continuously. For example, the first bytes (for example, $\{v_0^0, v_1^0, v_2^0, ...\}$) of a plurality of results are stored continuously to facilitate processing a plurality of results in parallel subsequently.

Table 1: Vector representation before transposition

| | $byte_0$ | $byte_1$ | $byte_2$ | ... |
|---|---|---|---|---|
| $v_0$ | $v_0^0$ | $v_0^1$ | $v_0^2$ | ... |
| $v_1$ | $v_1^0$ | $v_1^1$ | $v_1^2$ | ... |
| $v_2$ | $v_2^0$ | $v_2^1$ | $v_2^2$ | ... |
| ... | ... | ... | ... | ... |

Table 2. Vector representation after transposition

| | $v_0$ | $v_1$ | $v_2$ | ... |
|---|---|---|---|---|
| $byte_0$ | $v_0^0$ | $v_1^0$ | $v_2^0$ | ... |
| $byte_1$ | $v_0^1$ | $v_1^0$ | $v_2^1$ | ... |
| $byte_2$ | $v_0^2$ | $v_2^1$ | $v_2^2$ | ... |
| ... | ... | ... | ... | ... |

**[0055]** Fig. 6 shows a flowchart of a method 600 for determining a plurality of outputs for a plurality of inputs in parallel according to implementations of the subject matter described herein. It is to be understood that actions 602-608 in method 600 may be performed after action 206 in method 200 described with reference to Fig. 2 or action 414 described with reference to Fig. 4. It is to be understood that method 600 may be performed by the processing unit 110 described with reference to Fig. 1.

**[0056]** At 602, a plurality of starting bytes where the second bit value (such as 1) occurs in the transposed plurality of results are determined, and a plurality of indexes of a plurality of starting bytes are determined at 604. Next, at 606, a plurality of starting bit positions where the second bit value occurs in a plurality of starting bytes are determined. At 608, a plurality of outputs for a plurality of inputs are determined based on a plurality of indexes of a plurality of starting bytes and a plurality of starting bit positions. Since the position of the starting second bit value in each result may be used to determine the position of the leaf node, a plurality of outputs for a plurality of inputs may be determined by determining the position of the second bit value.

**[0057]** For example, Fig. 7 shows a schematic diagram 700 for determining a plurality of outputs for a plurality of inputs in parallel according to implementations of the subject matter described herein. The data shown in Fig. 7 is hexadecimal. For example, the decision tree model 175 generates $m$ results $\{v_0, v_1, v_2 \ldots v_{m-1}\}$ and then the vectors of the $m$ results are transposed. A plurality of transposed results 710 are generated, where a plurality of nodes at the same position in a plurality of results are stored continuously, for instance, a first byte 03 in $v_0$, a first byte 00 in $v_1$ and a first byte 00 in $v_2$...and a first byte 5B in $v_{m-1}$ are stored continuously.

**[0058]** The first non-zero bytes 720 in the plurality of results may be determined in parallel, for example, the data sequence (03, 00, 00,...5B) is firstly obtained, and the data sequence (4A, 00, 69,...72) is then obtained, and so on. Meanwhile, indexes $i_1$ 730 of the first non-zero bytes are determined. Next, indexes $i_2$ 740 of the first non-zero bit in the first non-zero byte 720 is determined. Then, based on the index $i_1$ 730 and the index $i_2$ 740, a plurality of indexes $i_0$ 750 (such as $i_0=8*i_1+i_2$) of the first non-zero bit in a plurality of result sequences are determined. The positions of a plurality of leaf nodes are determined based on the indexes $i_0$ of the first non-zero bit. In this way, a plurality of outputs for a plurality of inputs may be determined.

**[0059]** According to implementations of the subject matter described herein, by use of the vector transposition, the cache can be utilized such that data to be processed are stored continuously, thereby improving processing speed. In addition, the subject matter described herein uses the bit sequence to process a plurality of inputs in parallel and enables parallel data processing throughout the whole process in the decision tree model. Therefore, implementations of the subject matter described herein enable generation of a plurality of outputs for a plurality of inputs at a time, thereby improving processing efficiency of the decision tree.

**[0060]** Fig. 8 shows a schematic block diagram for a search system 800 for search ranking according to implementations of the subject matter described herein. As shown in Fig. 8, the search system 800 includes a document sorter 810 and a document database 820, and the document sorter 810 is used to rank the correlation matching between the received user query 830 and a plurality of documents in the document database 820. The document database 820 includes a plurality of documents such as webpages, document files, image files and the like. The document sorter 810 may generate N (for example, two or more) query-document pairs for the query 830, also referred to as (Q, D) pairs, and each (Q, D) pair has its respective feature set, such as a plurality of features $\{f_0, f_1, f_2, \ldots\}$, as described

above.

**[0061]** As shown in Fig. 8, the search system 800 further includes a decision tree model(s) 175 which includes a plurality of decision trees. The decision tree model 175 may receive feature sets of N(Q, D) pairs and process these feature sets with the method according to implementations of the subject matter described herein so as to generate *N* output scores 840. For example, the score of the first document may be 2.0, the score of second document may be 1.4 and so on. Then, the search system 800 can re-sort the scores 840 and output the document list to the user according to the re-sorted result. Therefore, the search system according to implementations of the subject matter described herein can enable generating the processing result efficiently by use of the decision tree model.

**[0062]** The method and functionally described herein may be performed, at least in part, by one or more hardware logic components. For example, and without limitation, illustrative types of hardware logic components that may be used include Field-Programmable Gate Arrays (FPGAs), Application-specific Integrated Circuits (ASICs), Application-specific Standard Products (ASSPs), System-on-a-chip systems (SOCs), Complex Programmable Logic Devices (CPLDs), and the like.

**[0063]** Program codes for carrying out methods of the subject matter described herein may be written in any combination of one or more programming languages. These program codes may be provided to a processor or controller of a general purpose computer, special purpose computer, or other programmable data processing apparatus, such that the program codes, when executed by the processor or controller, cause the functions/operations specified in the flowcharts and/or block diagrams to be implemented. The program codes may be executed entirely on a machine, partly on the machine, as a stand-alone software package partly on the machine and partly on a remote machine or entirely on the remote machine or server.

**[0064]** In the context of this subject matter described herein, a machine readable medium may be any tangible medium that may contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device. The machine readable medium may be a machine readable signal medium or a machine readable storage medium. A machine readable medium may include, but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, or device, or any suitable combination of the foregoing. More specific examples of the machine readable storage medium would include an electrical connection based on one or more wires, a portable

computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fiber, a portable compact disc read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing.

**[0065]** Further, while operations are depicted 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. Likewise, while several specific implementation details are contained in the above discussions, these should not be construed as limitations on the scope of the subject matter described herein, but rather as descriptions of features that may be specific to particular implementations. Certain features that are described in the context of separate implementations may also be implemented in combination in a single implementation. Conversely, various features that are described in the context of a single implementation may also be implemented in multiple implementations separately or in any suitable sub-combination.

**[0066]** Some example implementations of the subject matter described herein are listed in the following description.

**[0067]** In one aspect, there is provided an electronic device. The electronic device comprises a processing unit and a memory coupled to the processing unit and storing instructions. The instructions, when executed by the processing unit, perform acts including: obtaining a bit sequence including a first bit value and a second bit value; determining a starting position and an ending position where the first bit value occurs in the bit sequence; and encoding the bit sequence based on at least one of the starting position and the ending position.

**[0068]** In some implementations, wherein the bit sequence is divided into one or more bytes, and the determining a starting position and an ending position where the first bit value occurs in the bit sequence comprises: determining a first byte of the one or more bytes including the first bit value as a starting byte; determining an index of the starting byte as the starting position; determining a last byte of the one or more bytes including the first bit value as an ending byte; and determining an index of the ending byte as the ending position.

**[0069]** In some implementations, the encoding the bit sequence comprises: encoding the bit sequence based on starting byte, starting position, ending byte and ending position.

**[0070]** In some implementations, the encoding the bit sequence comprises: in response to the starting position being the same as the ending position, encoding the bit sequence based on the starting byte and the starting position.

**[0071]** In some implementations, wherein the bit sequence represents a feature vector of a node in a tree, and the node has a feature identification and an attribute value for determining a Boolean value of node, the Boolean value of node is determined based on a comparison between a feature value associated with the feature identification and the attribute value.

**[0072]** In some implementations, wherein the acts further comprise: determining a plurality of nodes in the tree which have both the feature identification and the attribute value; and combining operations for determining Boolean values of the plurality of nodes.

**[0073]** In some implementations, wherein the acts further comprise: sorting a plurality of attribute values of a plurality of nodes which have the feature identification in an ascending order; and in response to a feature value associated with the feature identification being below a particular attribute value of the plurality of attribute values, terminating a comparison between the feature value and an attribute value of the plurality of attribute values that is above the particular attribute value.

**[0074]** In some implementations, wherein the acts further comprise: receiving a plurality of inputs, wherein one of the plurality of inputs includes one or more feature values; and processing, for the node, the plurality of inputs in parallel.

**[0075]** In some implementations, wherein the acts further comprise: generating a plurality of results for the plurality of inputs by determining Boolean values of nodes in the tree; and transposing a vector representing the plurality of results so that a plurality of bytes in the plurality of results at the same position are continuously stored in a cache.

**[0076]** In some implementations, wherein the acts further comprise: determining a plurality of starting bytes where the second bit value occurs in the transposed plurality of results; determining a plurality of indexes of the plurality of starting bytes; determining a plurality of starting bit positions where the second bit value occurs in the plurality of starting bytes; and determining a plurality of outputs for the plurality of inputs based on the plurality of indexes of the plurality of starting bytes and the plurality of starting bit positions.

**[0077]** In another aspect, there is provided a computer-implemented method. The method includes: obtaining a bit sequence including a first bit value and a second bit value; determining a starting position and an ending position where the first bit value occurs in the bit sequence; and encoding the bit sequence based on at least one of the starting position

and the ending position.

**[0078]** In some implementations, wherein the bit sequence is divided into one or more bytes, and the determining a starting position and an ending position where the first bit value occurs in the bit sequence comprises: determining a first byte of the one or more bytes including the first bit value as a starting byte; determining an index of the starting byte as the starting position; determining a last byte of the one or more bytes including the first bit value as an ending byte; and determining an index of the ending byte as the ending position.

**[0079]** In some implementations, the encoding the bit sequence comprises: encoding the bit sequence based on starting byte, starting position, ending byte and ending position.

**[0080]** In some implementations, the encoding the bit sequence comprises: in response to the starting position being the same as the ending position, encoding the bit sequence based on the starting byte and the starting position.

**[0081]** In some implementations, wherein the bit sequence represents a feature vector of a node in a tree, and the node has a feature identification and an attribute value for determining a Boolean value of node, the Boolean value of node being determined based on a comparison between a feature value associated with the feature identification and the attribute value.

**[0082]** In some implementations, the method further comprises: determining a plurality of nodes in the tree which have both the feature identification and the attribute value; and combining operations for determining Boolean values of the plurality of nodes.

**[0083]** In some implementations, the method further comprises: sorting a plurality of attribute values of a plurality of nodes which have the feature identification in an ascending order; and in response to a feature value associated with the feature identification being below a particular attribute value of the plurality of attribute values, terminating a comparison between the feature value and an attribute value of the plurality of attribute values that is above the particular attribute value.

**[0084]** In some implementations, the method further comprises: receiving a plurality of inputs, wherein one of the plurality of inputs includes one or more feature values; and processing, for the node, the plurality of inputs in parallel.

**[0085]** In some implementations, the method further comprises: generating a plurality of results for the plurality of inputs by determining Boolean values of nodes in the tree; and transposing a vector representing the plurality of results so that a plurality of bytes in the plurality of results at the same position are continuously stored in a cache.

**[0086]** In some implementations, the method further comprises: determining a plurality of starting bytes where the second bit value occurs in the transposed plurality of results; determining a plurality of indexes of the plurality of starting bytes; determining a plurality of starting bit positions where the second bit value occurs in the plurality of starting bytes; and determining a plurality of outputs for the plurality of inputs based on the plurality of indexes of the plurality of starting bytes and the plurality of starting bit positions.

**[0087]** In another aspect, there is provided a computer program product. The computer program product is stored on a non-transient computer storage medium and comprises machine-executable instructions. The instructions, when executed on a device, cause the device to: obtain a bit sequence including a first bit value and a second bit value; determine a starting position and an ending position where the first bit value occurs in the bit sequence; and encode the bit sequence based on at least one of the starting position and the ending position.

**[0088]** In some implementations, wherein the bit sequence is divided into one or more bytes, and the determining a starting position and an ending position where the first bit value occurs in the bit sequence comprises: determining a first byte of the one or more bytes including the first bit value as a starting byte; determining an index of the starting byte as the starting position; determining a last byte of the one or more bytes including the first bit value as an ending byte; and determining an index of the ending byte as the ending position.

**[0089]** In some implementations, the encoding the bit sequence comprises: encoding the bit sequence based on starting byte, starting position, ending byte and ending position.

**[0090]** In some implementations, the encoding the bit sequence comprises: in response to the starting position being the same as the ending position, encoding the bit sequence based on the starting byte and the starting position.

**[0091]** In some implementations, wherein the bit sequence represents a feature vector of a node in a tree, and the node has a feature identification and an attribute value for determining a Boolean value of node, the Boolean value of node is determined based on a comparison between a feature value associated with the feature identification and the attribute value.

**[0092]** In some implementations, the machine-executable instructions, when executed on a device, further cause the device to: determine a plurality of nodes in the tree which have both the feature identification and the attribute value; and combine operations for determining Boolean values of the plurality of nodes.

**[0093]** In some implementations, the machine-executable instructions, when executed on a device, further cause the device to: sort a plurality of attribute values of a plurality of nodes which have the feature identification in an ascending order; and in response to a feature value associated with the feature identification being below a particular attribute value of the plurality of attribute values, terminate a comparison between the feature value and an attribute value of the plurality of attribute values that is above the particular attribute value.

**[0094]** In some implementations, the machine-executable instructions, when executed on the device, further cause the device to: receive a plurality of inputs, wherein one of the plurality of inputs includes one or more feature values; and process, for the node, the plurality of inputs in parallel.

**[0095]** In some implementations, the machine-executable instructions, when executed on a device, further cause the device to: generate a plurality of results for the plurality of inputs by determining Boolean values of nodes in the tree; and transpose a vector representing the plurality of results so that a plurality of bytes in the plurality of results at the same position are continuously stored in a cache.

**[0096]** In some implementations, the machine-executable instructions, when executed on a device, further cause the device to: determine a plurality of starting bytes where the second bit value occurs in the transposed plurality of results; determine a plurality of indexes of the plurality of starting bytes; determine a plurality of starting bit positions where the second bit value occurs in the plurality of starting bytes; and determine a plurality of outputs for the plurality of inputs based on the plurality of indexes of the plurality of starting bytes and the plurality of starting bit positions.

**[0097]** Although the subject matter described herein has been described with languages specific to structural characteristics and/or method logic actions, it should be appreciated that the subject matter defined by the appended claims is not limited to the above described particular characteristics and actions. Instead, the above described particular characteristics and actions are only example forms for realizing the claims.

## CLAIMS

1. An electronic device comprising:

a processing unit; and

a memory coupled to the processing unit and storing instructions that, when executed by the processing unit, perform acts including:

obtaining a bit sequence including a first bit value and a second bit value;

determining a starting position and an ending position where the first bit value occurs in the bit sequence; and

encoding the bit sequence based on at least one of the starting position and the ending position.

2. The device according to claim 1, wherein the bit sequence is divided into one or more bytes, and the determining a starting position and an ending position where the first bit value occurs in the bit sequence comprises:

determining a first byte of the one or more bytes including the first bit value as a starting byte;

determining an index of the starting byte as the starting position;

determining a last byte of the one or more bytes including the first bit value as an ending byte; and

determining an index of the ending byte as the ending position.

3. The device according to claim 2, wherein the encoding the bit sequence comprises:

encoding the bit sequence based on the starting byte, the starting position, the ending byte and the ending position.

4. The device according to claim 2, wherein the encoding the bit sequence comprises:

in response to the starting position being the same as the ending position, encoding the bit sequence based on the starting byte and the starting position.

5. The device according to claim 1, wherein the bit sequence represents a feature vector of a node in a tree, and the node has a feature identification and an attribute value for determining a Boolean value of node, the Boolean value of node being determined based on a comparison between a feature value associated with the feature identification and the attribute value.

6. The device according to claim 5, wherein the acts further comprise:

determining a plurality of nodes in the tree which have both the feature identification and the attribute value; and

combining operations for determining Boolean values of the plurality of nodes.

7. The device according to claim 5, wherein the acts further comprise:

sorting a plurality of attribute values of a plurality of nodes which have the feature identification in an ascending order; and

in response to a feature value associated with the feature identification being below a particular attribute value of the plurality of attribute values, terminating a comparison between the feature value and an attribute value of the plurality of attribute values that is above the particular attribute value.

8. The device according to claim 5, wherein the acts further comprise:

receiving a plurality of inputs, one of the plurality of inputs including one or more feature values; and

processing, for the node, the plurality of inputs in parallel.

9. The device according to claim 8, wherein the acts further comprise:

generating a plurality of results for the plurality of inputs by determining Boolean values of nodes in the tree; and

transposing a vector representing the plurality of results so that a plurality of bytes in the plurality of results at the same position are continuously stored in a cache.

10. The device according to claim 9, wherein the acts further comprise:

determining a plurality of starting bytes where the second bit value occurs in the transposed plurality of results;

determining a plurality of indexes of the plurality of starting bytes;

determining a plurality of starting bit positions where the second bit value occurs in the plurality of starting bytes; and

determining a plurality of outputs for the plurality of inputs based on the plurality of indexes of the plurality of starting bytes and the plurality of starting bit positions.

11. A computer-implemented method, comprising:

obtaining a bit sequence including a first bit value and a second bit value;

determining a starting position and an ending position where the first bit value occurs in the bit sequence; and

encoding the bit sequence based on at least one of the starting position and the ending position.

12. The method according to claim 11, wherein the bit sequence is divided into one or more bytes, and the determining a starting position and an ending position where the first bit value occurs in the bit sequence comprises:

determining a first byte of the one or more bytes including the first bit value as a starting byte;

determining an index of the starting byte as the starting position;

determining a last byte of the one or more bytes including the first bit value as an ending byte; and

determining an index of the ending byte as the ending position.

13. The method according to claim 12, wherein the encoding the bit sequence comprises:

encoding the bit sequence based on the starting byte, the starting position, the ending byte and the ending position.

14. The method according to claim 12, wherein the encoding the bit sequence comprises:

in response to the starting position being the same as the ending position, encoding the bit sequence based on the starting byte and the starting position.

15. A computer program product stored on a non-transient computer storage medium and comprising machine-executable instructions that, when executed on a device, cause the device to:

obtain a bit sequence including a first bit value and a second bit value;

determine a starting position and an ending position where the first bit value occurs in the bit sequence; and

encode the bit sequence based on at least one of the starting position and the ending position.

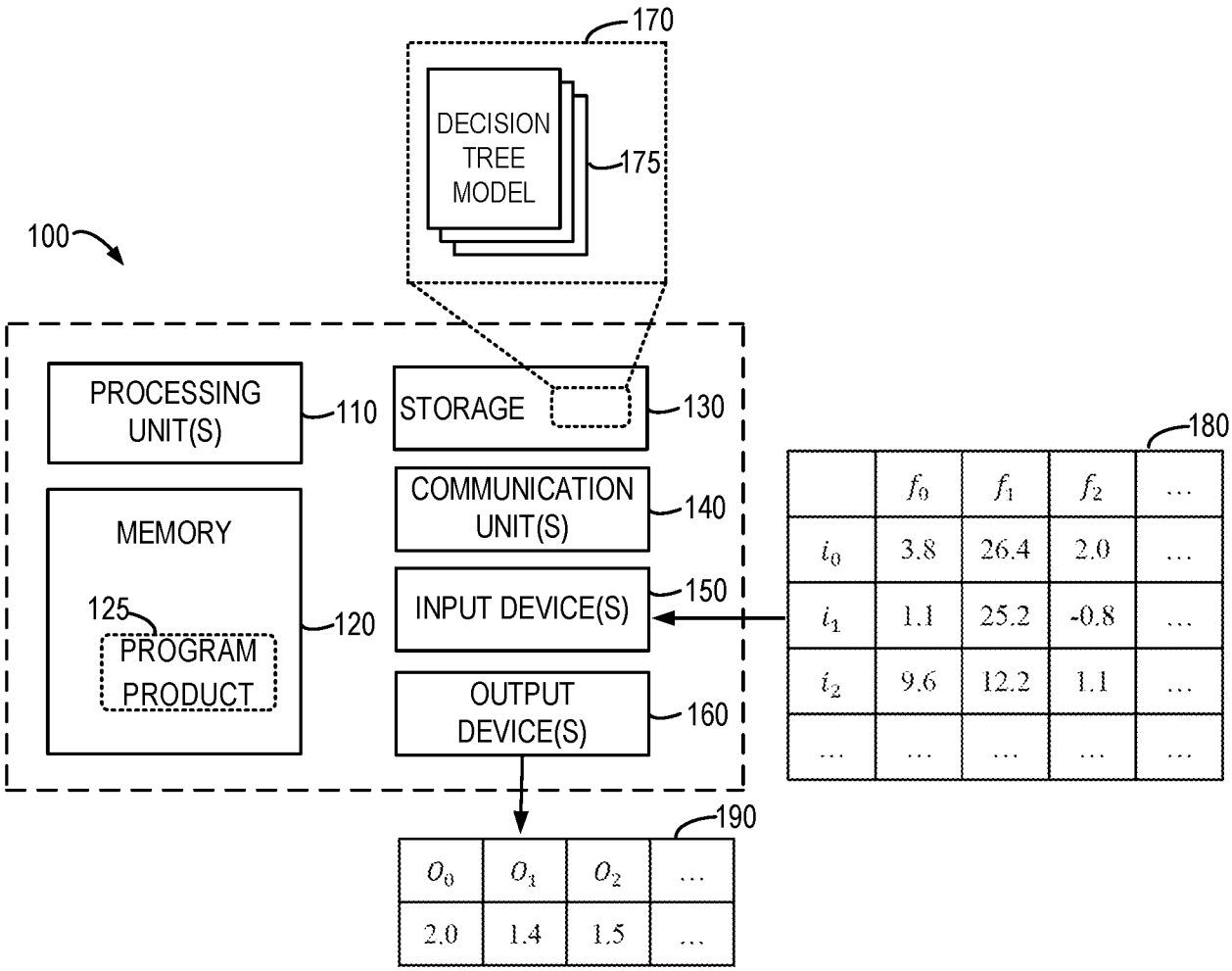
170
DECISION TREE MODEL
175
100
PROCESSING UNIT(S)
110
STORAGE
130
COMMUNICATION UNIT(S)
140
MEMORY
125
PROGRAM PRODUCT
120
INPUT DEVICE(S)
150
OUTPUT DEVICE(S)
160
180
$f_0$ $f_1$ $f_2$ …
$i_0$ 3.8 26.4 2.0 …
$i_1$ 1.1 25.2 -0.8 …
$i_2$ 9.6 12.2 1.1 …
… … … … …
190
$O_0$ $O_1$ $O_2$ …
2.0 1.4 1.5 …

FIG. 1

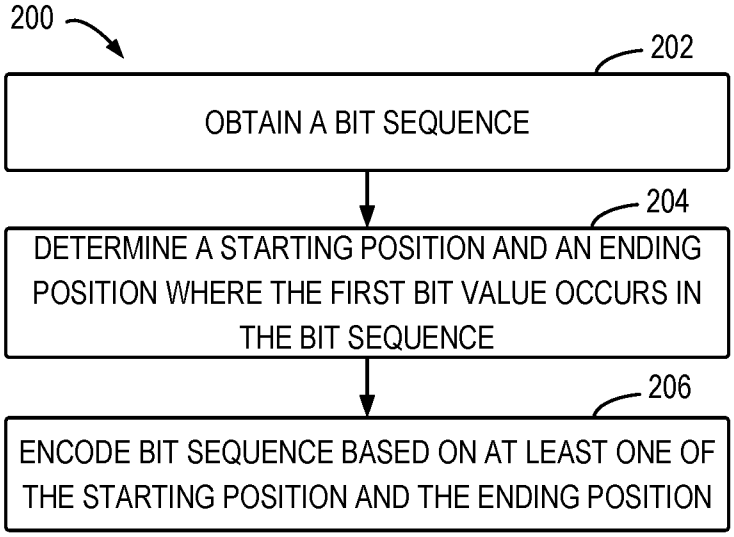
200
202
OBTAIN A BIT SEQUENCE
204
DETERMINE A STARTING POSITION AND AN ENDING POSITION WHERE THE FIRST BIT VALUE OCCURS IN THE BIT SEQUENCE
206
ENCODE BIT SEQUENCE BASED ON AT LEAST ONE OF THE STARTING POSITION AND THE ENDING POSITION

FIG. 2

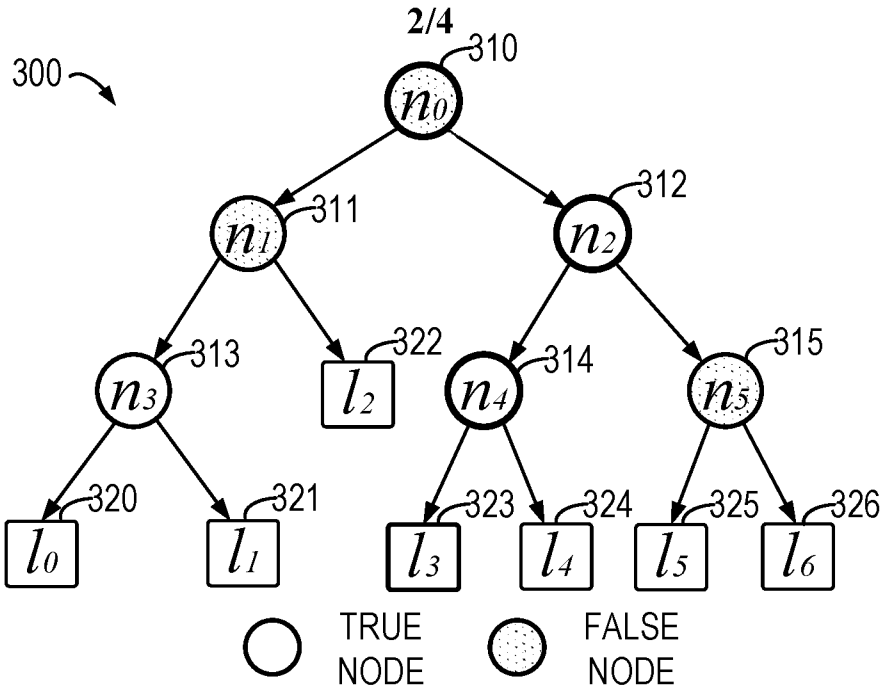
300
310
$n_0$
311
$n_1$
312
$n_2$
313
$n_3$
322
$l_2$
314
$n_4$
315
$n_5$
320
$l_0$
321
$l_1$
323
$l_3$
324
$l_4$
325
$l_5$
326
$l_6$
TRUE NODE
FALSE NODE

FIG. 3

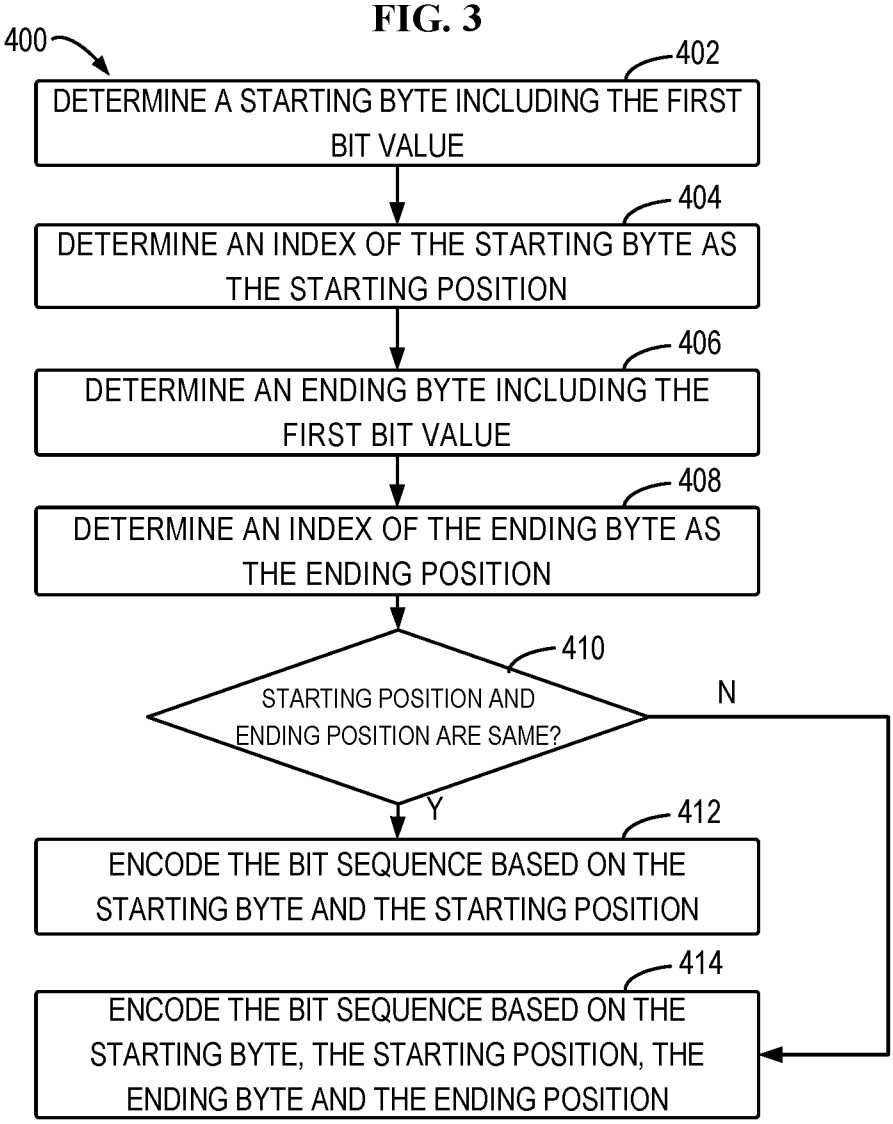
400
402
DETERMINE A STARTING BYTE INCLUDING THE FIRST BIT VALUE
404
DETERMINE AN INDEX OF THE STARTING BYTE AS THE STARTING POSITION
406
DETERMINE AN ENDING BYTE INCLUDING THE FIRST BIT VALUE
408
DETERMINE AN INDEX OF THE ENDING BYTE AS THE ENDING POSITION
410
STARTING POSITION AND ENDING POSITION ARE SAME?
N
Y
412
ENCODE THE BIT SEQUENCE BASED ON THE STARTING BYTE AND THE STARTING POSITION
414
ENCODE THE BIT SEQUENCE BASED ON THE STARTING BYTE, THE STARTING POSITION, THE ENDING BYTE AND THE ENDING POSITION

FIG. 4

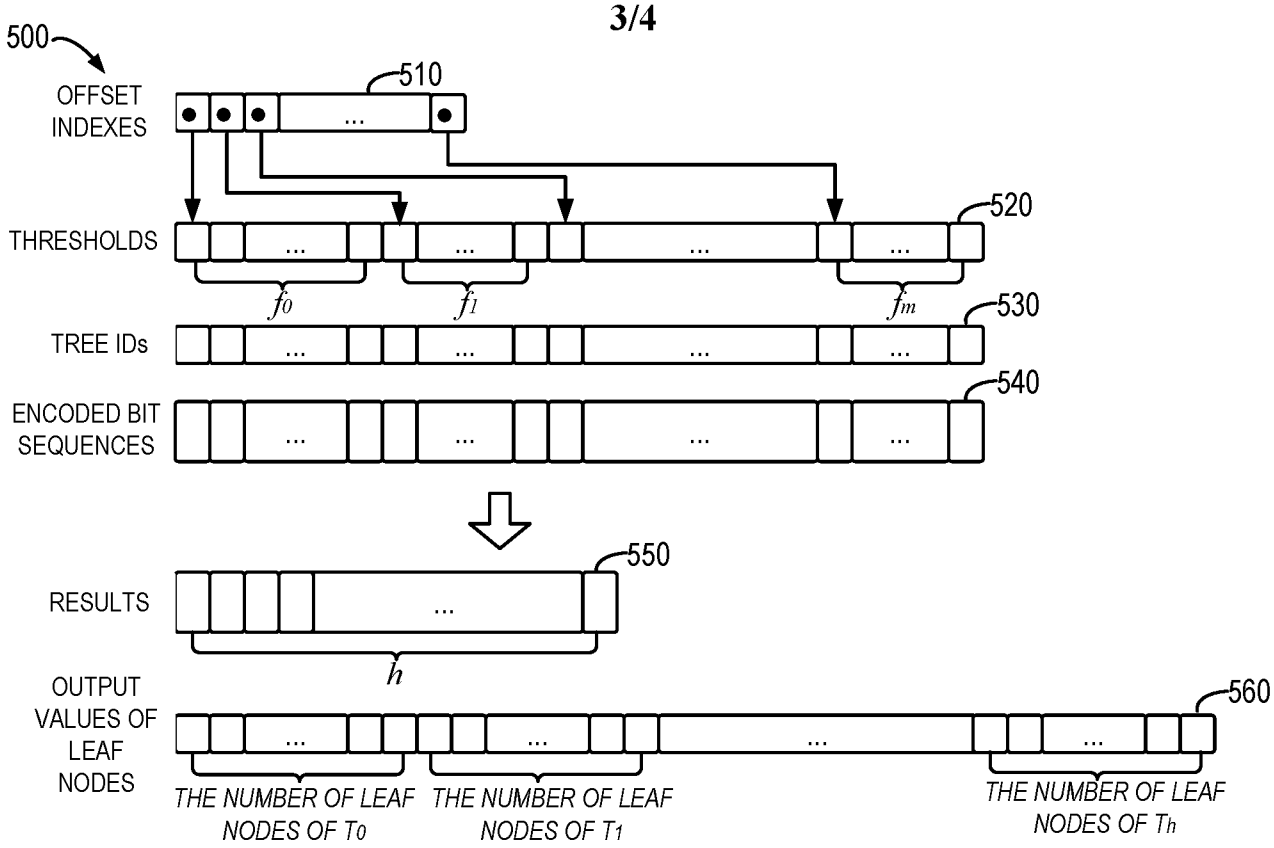
500
OFFSET INDEXES
510
THRESHOLDS
520
$f_0$
$f_1$
$f_m$
TREE IDs
530
ENCODED BIT SEQUENCES
540
RESULTS
550
$h$
OUTPUT VALUES OF LEAF NODES
560
THE NUMBER OF LEAF NODES OF $T_0$
THE NUMBER OF LEAF NODES OF $T_1$
THE NUMBER OF LEAF NODES OF $T_h$

FIG. 5

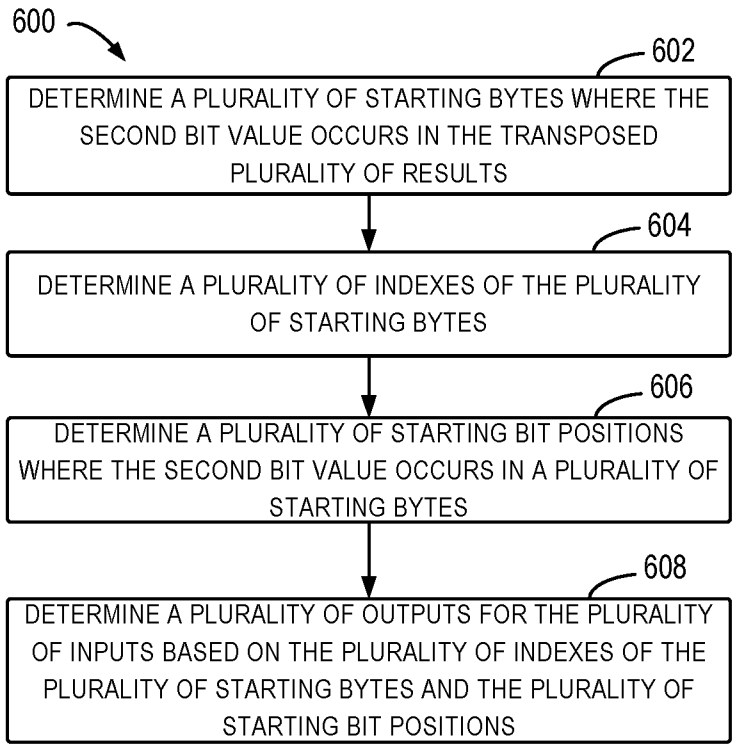
600
602
DETERMINE A PLURALITY OF STARTING BYTES WHERE THE SECOND BIT VALUE OCCURS IN THE TRANSPOSED PLURALITY OF RESULTS
604
DETERMINE A PLURALITY OF INDEXES OF THE PLURALITY OF STARTING BYTES
606
DETERMINE A PLURALITY OF STARTING BIT POSITIONS WHERE THE SECOND BIT VALUE OCCURS IN A PLURALITY OF STARTING BYTES
608
DETERMINE A PLURALITY OF OUTPUTS FOR THE PLURALITY OF INPUTS BASED ON THE PLURALITY OF INDEXES OF THE PLURALITY OF STARTING BYTES AND THE PLURALITY OF STARTING BIT POSITIONS

FIG. 6

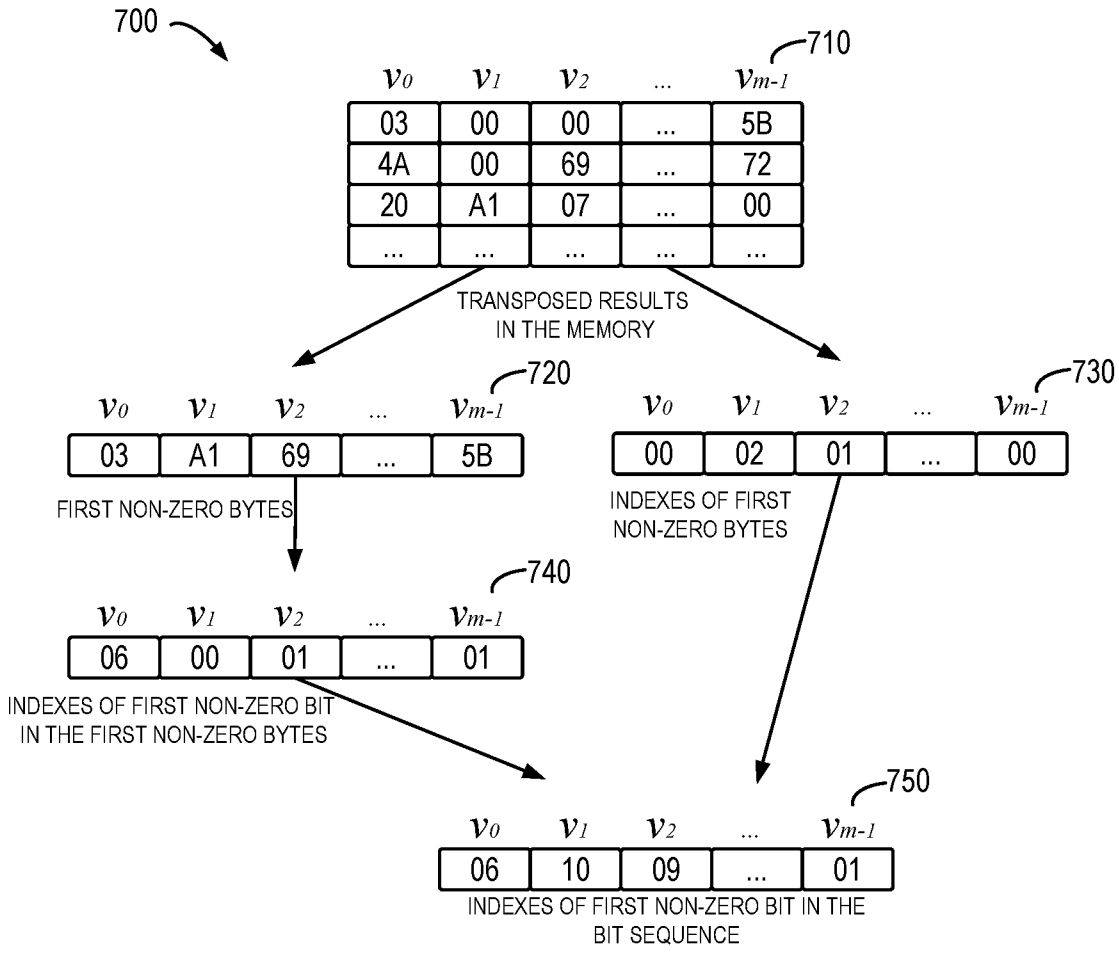
700
710
$v_0$ $v_1$ $v_2$ ... $v_{m-1}$
03 00 00 ... 5B
4A 00 69 ... 72
20 A1 07 ... 00
... ... ... ... ...
TRANSPOSED RESULTS IN THE MEMORY
720
$v_0$ $v_1$ $v_2$ ... $v_{m-1}$
03 A1 69 ... 5B
FIRST NON-ZERO BYTES
730
$v_0$ $v_1$ $v_2$ ... $v_{m-1}$
00 02 01 ... 00
INDEXES OF FIRST NON-ZERO BYTES
740
$v_0$ $v_1$ $v_2$ ... $v_{m-1}$
06 00 01 ... 01
INDEXES OF FIRST NON-ZERO BIT IN THE FIRST NON-ZERO BYTES
750
$v_0$ $v_1$ $v_2$ ... $v_{m-1}$
06 10 09 ... 01
INDEXES OF FIRST NON-ZERO BIT IN THE BIT SEQUENCE

FIG. 7

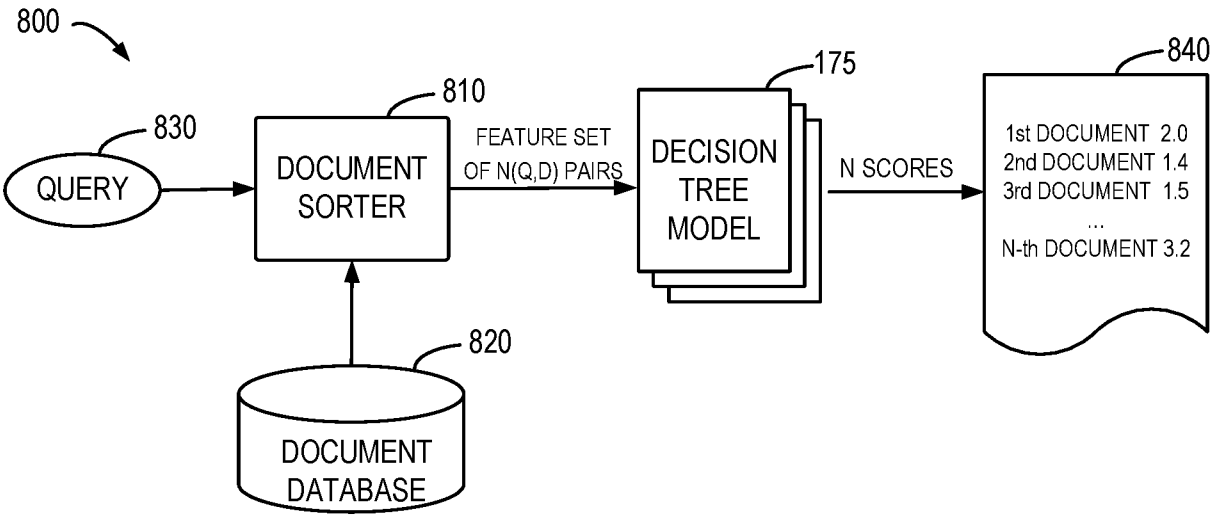
800
830
QUERY
810
DOCUMENT SORTER
FEATURE SET OF N(Q,D) PAIRS
175
DECISION TREE MODEL
N SCORES
840
1st DOCUMENT 2.0
2nd DOCUMENT 1.4
3rd DOCUMENT 1.5
...
N-th DOCUMENT 3.2
820
DOCUMENT DATABASE

FIG. 8

# INTERNATIONAL SEARCH REPORT

International application No
PCT/US2018/013750

A. CLASSIFICATION OF SUBJECT MATTER
INV. H03M7/46
ADD.

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
H03M

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

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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | "5.5 Data Compression", internet article, 27 October 2016 (2016-10-27), XP002779836, Retrieved from the Internet: URL:https://www.cs.princeton.edu/courses/archive/spr10/cos226/lectures/18-55DataCompression-2x2.pdf [retrieved on 2018-04-09] Slide 21 | 1-15 |

-----

-/--

[X] Further documents are listed in the continuation of Box C.

[ ] See patent family annex.

* Special categories of cited documents :

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

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

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

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

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

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

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

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

"&" document member of the same patent family

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 9 April 2018 | 02/05/2018 |

| Name and mailing address of the ISA/ | Authorized officer |
|---|---|
| European Patent Office, P.B. 5818 Patentlaan 2<br>NL - 2280 HV Rijswijk<br>Tel. (+31-70) 340-2040,<br>Fax: (+31-70) 340-3016 | Rydyger, Kay |

# INTERNATIONAL SEARCH REPORT

International application No
PCT/US2018/013750

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | Thinh Nguyen: "Lecture 7:Run-Length, Golomb, and Tunstall Codes", internet artile, 22 August 2006 (2006-08-22), XP002779837, Retrieved from the Internet: URL:http://web.engr.oregonstate.edu/~thinhq/teaching/ece499/spring06/runlength_turnstall_golomb.pdf [retrieved on 2018-04-09] the whole document<br>----- | 1-15 |