Science Advances

AAAS

advances.sciencemag.org/cgi/content/full/2/5/e1600028/DC1

Supplementary Materials for

Representing higher-order dependencies in networks

Jian Xu, Thanuka L. Wickramarathne, Nitesh V. Chawla

Published 20 May 2016, *Sci. Adv.* **2**, e1600028 (2016)
DOI: 10.1126/sciadv.1600028

The PDF file includes:

note S1. Data sets

Global shipping data. This data made available by Lloyd's Maritime Intelligence Unit (LMIU) contains ship movement information such as `vessel_id`, `port_id`, `sail_date` and `arrival_date`. Our experiments are based on a recent LMIU data set that spans one year from May 1st, 2012 to April 30th, 2013, totaling 3,415,577 individual voyages corresponding to 65,591 ships that move among 4,108 ports and regions globally. A minimum support of 10 is used to filter out noise in the data.

Clickstream data. This data made available by a media company contains logs of users clicking through web pages that belong to 50 news web sites owned by the company. Fields of interest include `user_ip`, `pagename` and `time`. Our experiments are based on the clickstream records that span two months from December 4th, 2012 to February 3rd, 2013, totaling 3,047,697 page views made by 179,178 distinct IP addresses on 45,257 web pages. A minimum support of 5 is used to filter out noise in the data. Clickstreams that are likely to be created by crawlers (abnormally long clickstreams / clickstreams that frequently hit the error page) are omitted.

Retweet data. This data (*50*) records retweet history on Weibo (a Chinese microblogging website), with information about who retweets whose messages at what time. The data was crawled in 2012 and there are 23,755,810 retweets recorded, involving 1,776,950 users.

Synthetic data. We created a trajectory data set (data and code are available at https://github.com/xyjprc/hon) with known higher-order dependencies to verify the effectiveness of the rule extraction algorithm. In the context of shipping, we connect 100 ports as a 10×10 grid, then generate trajectories of 100,000 ships moving among these ports. Each ship moves 100 steps, yielding 10,000,000 movements in total. Normally each ship has equal probabilities of going up/down/left/right on the grid in each step (with wrapping, e.g., going down at the bottom row will end up in the top row); we use

additional higher-order rules to control the generation of ship movements. For example, a second-order rule can be defined as whenever a ship comes from Shanghai to Singapore, instead of randomly picking a neighboring port of Singapore for the next step, the ship has 70% chance of going to Los Angeles and 30% chance of going to Seattle. We predefine 10 second-order rules like this, and similarly 10 third-order rules, 10 fourth-order rules, and no other higher-order rules, so that movements that have variable orders of dependencies are generated. To test the rule extraction algorithm, we set the maximum order as five to see if the algorithm will incorrectly extract false rules beyond the fourth order which we did not define; we set minimum support as five for patterns to be considered as rules.

note S2. Algorithms

Rule extraction. Algorithm 1 gives the pseudocode of rule extraction. The procedure consists of three major steps: BUILDOBSERVATIONS counts the frequencies of all subsequences from the second order to *MaxOrder* for every trajectory in *T* (fig. S1 ①); BUILDDISTRIBUTIONS first builds all paths by removing subsequences that appear less than *MinSupport* times, then estimates probability distributions of movements at every source node by normalizing the observed frequencies (fig. S1 ②). GENERATEALLRULES starts from the first order and tries to increase the order recursively, by including an additional entity at the beginning of the entity sequence of the source node and testing if the probability distribution for the next step changes significantly (fig. S1 ③).

The comparison between probability distributions is performed by the recursive function EXTENDRULE. *Curr* is the current source node (X_t = *Singapore* in fig. S1), which is to be extended into a higher-order source node *ExtSource* by including the previous step (X_t = *Singapore*, X_{t-1} = *Shanghai* in fig. S1) (line 40). *Valid* is the last known source node from which a random walker has significantly different probability distributions towards the next step, i.e., the last assumed order for the path starting from node *Valid* is length(*Valid*). If at the extended source node, a random walker has a significantly

different probability distribution of the next movement compared with that at node *Valid*, the extended source node will be marked as the new *Valid* for the recursive growing of rule (line 47), otherwise the old *Valid* is kept (line 49). The paths with *Valid* as the source node have correct orders of dependencies, and will be added to the rules set R whenever EXTENDRULE exceeds *MaxOrder* (line 35) or the source node cannot be extended (line 41, true when no higher-order source node with the same last steps exists). When a higher-order rule (a path from *Source*) is added, all paths of the preceding steps of *Source* are added (line 54–55) to ensure the network wiring step can connect nodes with variable orders. For example, when paths from the source node *Singapore*|*Shanghai* are added to *R*, the preceding step *Shanghai* → *Singapore* should also be added to *R*.

Specifically, to determine whether extending the source node from *Valid* to *ExtSource* significantly changes the probability distribution for the next step (line 46), we compute the Kullback–Leibler divergence (*51*) between the two distributions, since it is a widely-used and standard way of comparison probability distributions (*22–24*). We consider the change is significant if the divergence satisfies $Divergence_{KL}(P_{Valid}, P_{ExtSource}) > \frac{Order_{ExtSource}}{\log_2(Support_{ExtSource})}$, based on the following intuition, inspired by Ben-Gal 2005 (*24*) and Bühlmann 1999 (*22*): for the two extended nodes showing the same divergence with regard to the original source node, (1) we are more inclined to prune the node with higher orders (the one with more previous steps embedded); and (2) we are less inclined to prune the node with higher support (the one with more trajectories going through). Instead of applying a universal threshold for all nodes that may have varying orders and supports, the threshold we adopt is self-adjustable for different nodes.

It is worth noting that Algorithm 1 also applies to data types other than trajectories such as diffusion data, which needs only one change of the EXTRACTSUBSEQUENCES function such that it takes only the newest entity subsequence. In addition, although higher-order

dependencies exist in many types of data, it is the type of data (vessel trajectory data / gene sequence data / language data / diffusion data …) that determines whether there are higher-order dependencies and how high the orders can be. Our proposed algorithm is backwards compatible with data that have no higher-order dependencies (such as the diffusion data): all rules extracted are first-order, thus the output will be a first-order network.

Network wiring. Algorithm 2 gives the pseudocode for converting rules extracted from data into a graph representation. In line 3, sorting rules by the length of key *Source* is equivalent to sorting rules by ascending orders. This ensures that the for loop in line 4–7 converts all lower order rules before processing higher-order rules. For every order, line 5 converts rules to edges, and line 7 REWIRE(*r*) attempts rewiring if it is not the first order. Figure S2C illustrates the case *r* = *Singapore*|*Shanghai* → *Los Angeles* in line 11, indicating *PrevSource* is *Shanghai* and *PrevTarget* is *Singapore*|*Shanghai*. The edge *Shanghai* → *Singapore*|*Shanghai* is not found in the network, so in line 15 and 16 *Shanghai* → *Singapore*|*Shanghai* replaces *Shanghai* → *Singapore* using the same edge weight.

After converting all rules in REWIRETAILS, edges created from *Valid* rules are rewired to target nodes that have the highest possible orders. Figure S2D illustrates the following case: in line 21 *r* = *Singapore*|*Shanghai* → *Seattle*, so in line 22 *NewTarget* is assigned as *Seattle*|*Singapore*,*Shanghai*; assume *Seattle*|*Singapore*,*Shanghai* is not found in all sources of *R*, so the lower order node *Seattle*|*Singapore* is searched next; assume *Seattle*|*Singapore* already exists in the graph, so *Singapore*|*Shanghai* → *Seattle*|*Singapore* replaces *Singapore*|*Shanghai* → *Seattle*.

note S3. Parameter discussion

Minimum support. As mentioned in the methods section, only when a subsequence occurs sufficiently many times (not below minimum support) can it be distinguished from noise and construed as a (non-trivial) path. While a minimum support is not compulsory, setting an appropriate minimum support can significantly reduce the network size and improve the accuracy of representation. As shown in fig. S3A, by increasing the minimum support from 1 to 10 (with a fixed maximum order of 5), the size of the network shrinks by 20 times while the accuracy of random walking simulation increases by 0.54%. By increasing the minimum support from 1 to 100, the accuracy first increases then decreases. The reason is that with low minimum support, some unusual subsequences that are noise are counted as paths; on the other hand, a high minimum support leaves out some true patterns that happen less frequently. The optimal minimum support (that can increase the accuracy of representation and greatly reduce the size of the network) may not be the same for different types of data, but can be found by parameter sweeping.

Maximum order. With a higher maximum order, the rule extraction algorithm can capture dependencies of higher orders, leading to higher accuracies of random walking simulations. As shown in fig. S3B, when increasing the maximum order from 1 to 5 (with a fixed minimum support of 10), the accuracy of random walking simulation keeps increasing but converges at the maximum order of 5, and the same trend applies for the size of the network. The reason is that the majority of dependencies have lower orders while fewer dependencies have higher orders, which again justifies our approach of not assigning a fixed high order for the whole network. On the other hand, setting a high maximum order does not significantly increase the running time of building HON, because in the rule extraction algorithm, most subsequences of longer lengths do not satisfy the minimum support requirement and are not considered in the following steps. In brief, when building HON using the aforementioned algorithm, an order that is sufficiently high can be assumed as the maximum order (a maximum order of five is sufficient for most applications).

note S4. Empirical comparison with the Variable-order Markov (VOM) model

In Materials and Methods we mentioned the differences between HON and the Variable-order Markov (VOM) model *(22–24)*. In this section we first illustrate these differences with an example, then provide an empirical comparison between HON and VOM. Note that because the "smoothing" process is not a compulsory step of VOM, we do not apply it in the following comparison, although "smoothing" is undesirable for network representation.

Example. In the context of global shipping, suppose ports *a*, *b*, *c*, …, *h*, *i* are connected as shown in fig. S4A. Port *f* and *g* are at the two ends of a canal. We assume that all ships coming from *d* through the canal will go to *h*, and all ships coming from *e* through the canal will go to *i*. A possible set of ship trajectories are listed in fig. S4B. Based on these trajectories, we can count the frequencies of subsequences, and compute the probability distributions of next steps given the previous ports visited (HON and VOM yield identical results). The subsequences of variable orders can naturally form a tree as shown in fig. S4C, where source nodes are in circles and target nodes (and the corresponding edge weights) are in the boxes below.

HON and VOM have different mechanisms of deciding which nodes to retain in the tree. In fig. S4C, the nodes kept by HON are denoted by red stars and nodes kept by VOM are denoted by purple triangles, showing a mismatch of the results. For HON, although *g|f* does not show second order dependency (having the same probability distribution with *g*), *g|f,d* shows third order dependency (having significantly different probability distribution compared with g), so *g|f,d* is retained by HON. According to AddToRules of the "rule extraction" step (Algorithm 1), all preceding nodes are retained, including *f|d* and *d*, such that the "network wiring" step already has exactly the nodes needed: there would be a path of $d \rightarrow f|d \rightarrow g|f,d$, as shown in fig. S4D. On the contrary, in the VOM construction process, after determining that *g|f,d* is a higher-order node to be kept, VOM keeps *g|f*, and prunes *f|d*, despite that (1) *f|d* is necessary for building the link to *g|f,d*

when constructing a network, and (2) $g|f$ is not necessary for building HON as it has the same probability distribution with g.

The eventual wiring of HON is shown in fig. S4D. Compared with the true connection in fig. S4A, HON not only keeps the first order links, but also adds higher-order nodes and edges for the two ports *f* and *g* in the canal, successfully capturing the pattern that "all ships coming from *d* through the canal will go to *h*, and all ships coming from *e* through the canal will go to *i*".

An additional difference between HON and VOM is how they determine the orders of rules. HON assumes the first order initially and compares with higher orders, while VOM "prunes" rules recursively from higher orders to lower orders, which as illustrated in fig. S4E, may prune higher-order nodes despite they have very different distributions than first-order nodes (e.g., $z|y,x,w$ compared with z), thus underestimating the orders of dependencies.

In brief, we have shown that VOM cannot be used directly to construct HON, given that VOM (1) retains unnecessary nodes for constructing HON, (2) prunes necessary nodes, and (3) has a pruning mechanism that may leave out certain higher-order dependencies.

Numerical comparison. To show the differences of HON and VOM quantitatively, we apply both HON and VOM to the same global shipping data set, assume the same filtering for preprocessing (*MaxOrder* = 5 and *MinSupport* = 10), use the same distance measure (KL divergence), and for fair comparison, we use the same threshold $\frac{Order_{ExtSource}}{\log_2(Support_{ExtSource})}$ for judging whether two distributions are significantly different. table S2 gives the comparison of the number of rules extracted from both algorithms.

We can observe that the rules extracted by HON and VOM show considerable differences except for the first order, even though these two algorithms are given the

same parameters. The different mechanisms of deciding which nodes to keep lead to the differences in the extracted rules. This further supports our claim that the rules extracted by VOM cannot be readily used for building HON, while the “rule extraction” process of HON has already prepared exactly the rules needed and only need to rewire some links.

Supplementary figures, tables, and algorithms

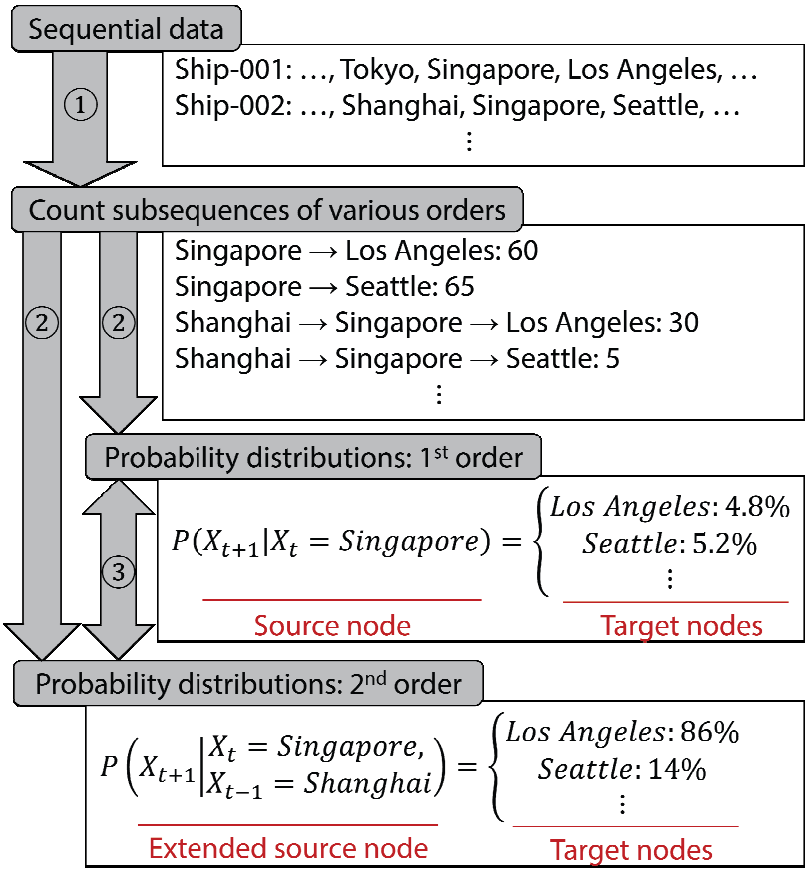

fig. S1. Rule extraction example for the global shipping data. Step 1: count the occurrences of subsequences from the first order to the maximum order, and keep those that meet the minimum support requirement. Step 2: given the source node representing a sequence of entities as the previous step(s), compute probability distributions for the next step. Step 3: given the original source node and an extended source node (extended by including an additional entity at the beginning of the entity sequence), compare the probability distributions of the next step. For example, when the current location is Singapore, knowing that a ship comes from Shanghai to Singapore (second order) significantly changes the probability distribution for the next step compared with not knowing where the ship came from (first order). So the second-order dependency is assumed here; then the probability distribution is compared with that of the third order, and so on, until the minimum support is not met or the maximum order is exceeded.

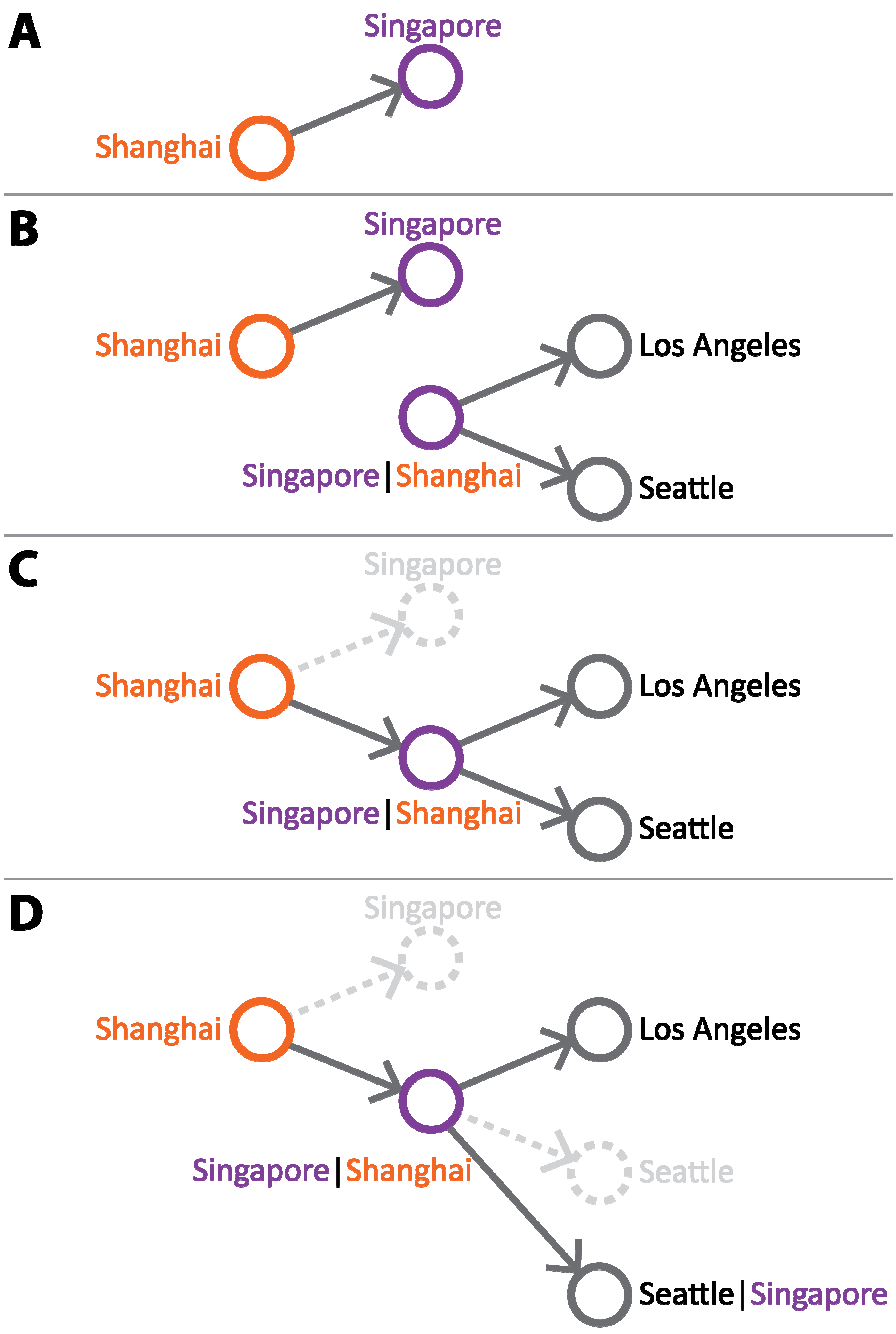

fig. S2. Network wiring example for the global shipping data. Figure shows how the dependency rules are represented as HON. (**A**) convert all first-order rules into edges; (**B**) convert higher-order rules, and add higher-order nodes when necessary, (**C**) rewire edges so that they point to newly added higher-order nodes (the edge weights are preserved); (**D**) rewire edges built from Valid rules so that they point to nodes with the highest possible order.

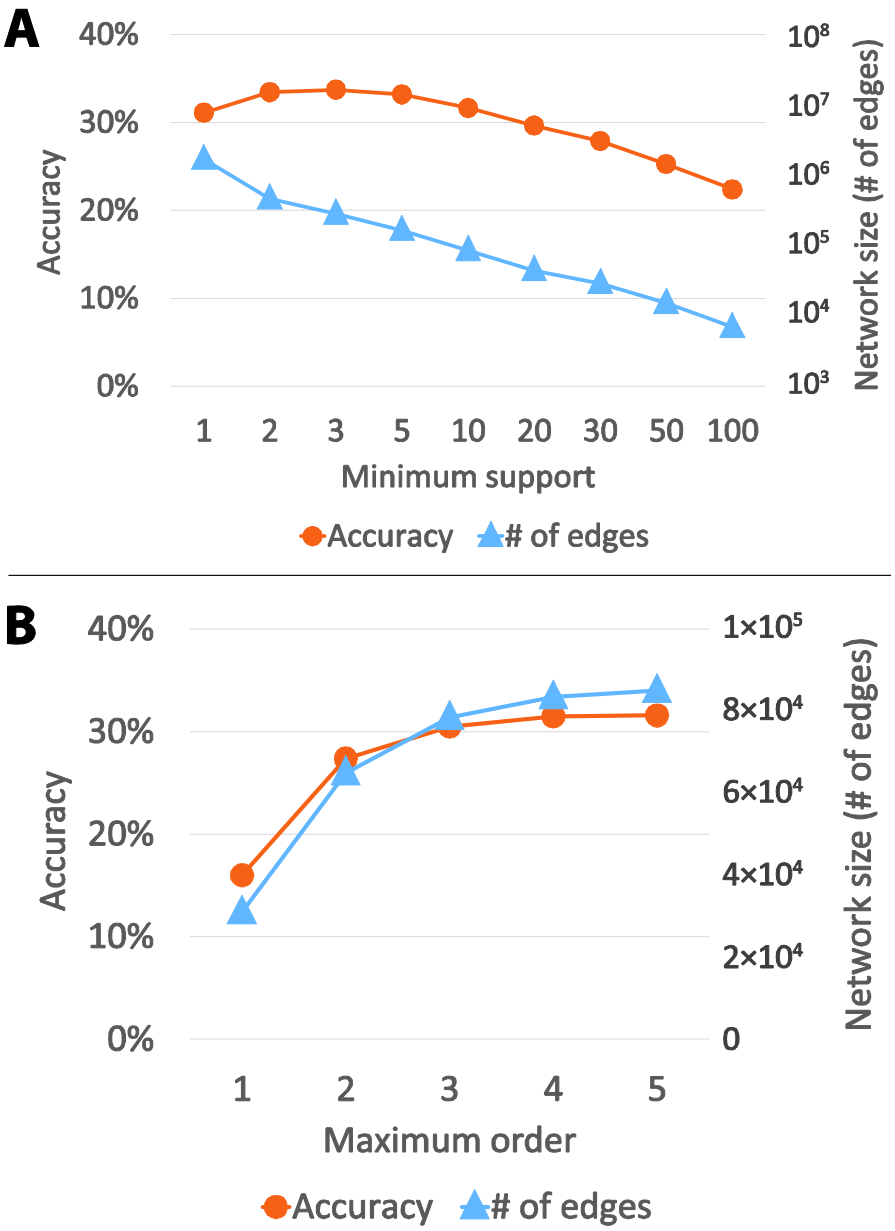

fig. S3. Parameter sensitivity of HON in terms of the accuracy and network size. The global shipping data is illustrated, and the accuracy is the percentage of correct predictions when using a random walker to predict the next step. (**A**) An appropriate minimum support can significantly reduce the network size and improve the accuracy of representation; (**B**) when increasing the maximum order, the accuracy of random walking simulation keeps improving but converges near the maximum order of 5.

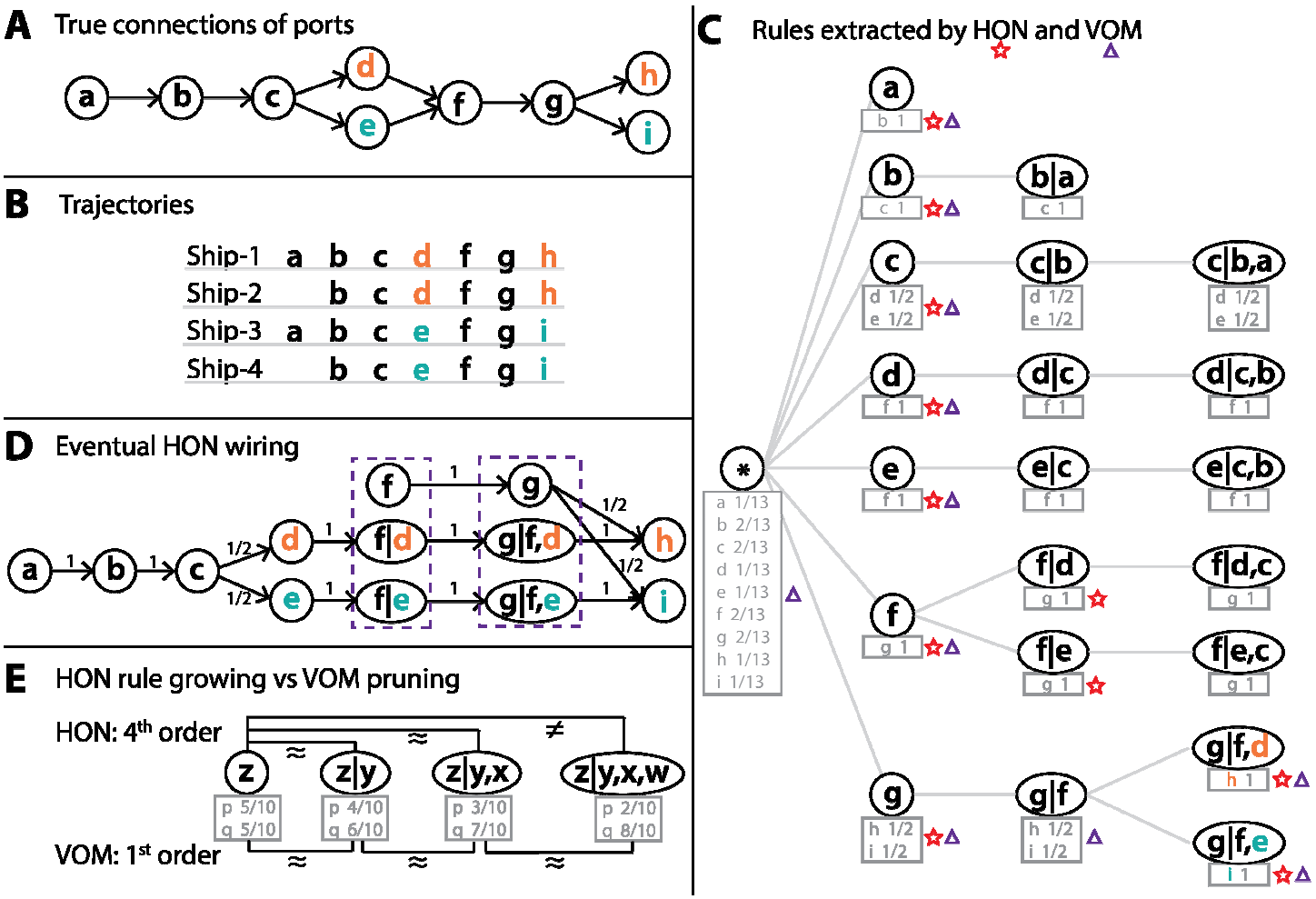

fig. S4. Comparison between the HON and the VOM model. (**A**) In the context of global shipping, the true connection of ports. *f* and *g* are at the two sides of a canal. Ships coming from *d* will go to *h*, and coming from *e* will go to *i*. (**B**) Possible trajectories of ships. (**C**) Comparing the nodes retained by HON and VOM. VOM prunes nodes that are necessary for network representation while retaining nodes that are not necessary. (**D**) The eventual HON representation captures higher-order dependencies while retaining all first-order information. (**E**) HON "grows" rules from the first order, while VOM prunes rules from the highest order.

table S1. Changes of PageRank scores by using HON instead of a first-order network. For the clickstream data. Top 15 gainers / losers of PageRanks scores are listed.

Pages that gain PageRank scores	ΔPageRank
South Bend Tribune - Home.	+0.0119
Hagerstown News / obituaries - Front.	+0.0115
South Bend Tribune - Obits - 3rd Party.	+0.0112
South Bend Tribune / sports / notredame - Front.	+0.0102
Aberdeen News / news / obituaries - Front.	+0.0077
WDBJ7 - Home.	+0.0075
KY3 / weather - Front.	+0.0075
Hagerstown News - Home.	+0.0072
Daily American / lifestyle / obituaries - Front.	+0.0054
WDBJ7 / weather / closings - Front.	+0.0048
WSBT TV / weather - Front.	+0.0041
Daily American - Home.	+0.0036
WDBJ7 / weather / radar - Front.	+0.0036
WDBJ7 / weather / 7-day-planner - Front.	+0.0031
WDBJ7 / weather - Front.	+0.0019
Pages that lose PageRank scores	**ΔPageRank**
KTUU - Home.	−0.0057
KWCH - Home.	−0.0031
Imperial Valley Press - Home.	−0.0011
Hagerstown News / sports - Front.	−0.0005
Imperial Valley Press / classifieds / topjobs - Front.	−0.0004
Gaylord - Home.	−0.0004
WDBJ7 / weather / web-cams - Front.	−0.0004
KTUU / about / meetnewsteam - Front.	−0.0003
Smithsburg man faces more charges following salvag - Hagerstown News / news - story.	−0.0003
KWCH / about / station / newsteam - Front.	−0.0003
South Bend Tribune / sports / highschoolsports - Front.	−0.0003
Hagerstown News / opinion - Front.	−0.0002
WDBJ7 / news / anchors-reporters - Front.	−0.0002
Petoskey News / news / obituaries - Front.	−0.0002
KWCH / news - Front.	−0.0002

table S2. **Comparison of the number of rules extracted from HON and VOM**. Both algorithms are applied on the same global shipping data set, with the same parameters.

	HON	**VOM**	**In HON but not in VOM**	**In VOM but not in HON**
0^{th} order	0	3,029	0	3029
1^{st} order	31,028	31,028	0	0
2^{nd} order	32,960	35,288	427	2,755
3^{rd} order	15,642	21,536	550	6,444
4^{th} order	4,632	8,973	302	4,643
5^{th} order	763	2,084	23	1,344
Total	85,025	101,938	1,302	18,215

algorithm S1. The rule extraction algorithm.

Algorithm 1 Rule extraction. Given the sequential data T, outputs higher-order dependency rules R.
Parameters: $\mathcal{O}$: MaxOrder, S: MinSupport

global counter $C \leftarrow \emptyset$
global nested dictionary $D \leftarrow \emptyset$
global nested dictionary $R \leftarrow \emptyset$

function EXTRACTRULES(T)
    BUILDOBSERVATIONS(T)
    BUILDDISTRIBUTIONS()
    GENERATEALLRULES()
    return R

function BUILDOBSERVATIONS(T)
    for t in T **do**
        for o from 2 to $\mathcal{O}$ **do**
            $SS \leftarrow$ ExtractSubSequences(t, o)
            for s in SS **do**
                $Target \leftarrow$ LastElement(s)
                $Source \leftarrow$ AllButLastElement(s)
                IncreaseCounter($C[Source][Target]$)

function BUILDDISTRIBUTIONS()
    for $Source$ in C **do**
        for $Target$ in $C[Source]$ **do**
            if $C[Source][Target] < S$ **then**
                Remove(C[Source][Target])
        for $Target$ in $C[Source]$ **do**
            $D[Source][Target] \leftarrow C[Source][Target]$ / Sum($C[Source][*]$)

function GENERATEALLRULES()
    for $Source$ in D **do**
        if length($Source$)$= 1$ **then**
            ADDTORULES($Source$)
            EXTENDRULE($Source, Source, 1$)

(To be continued on the next page.)

Algorithm 1 *(continued)*

function EXTENDRULE($Valid, Curr, Order$)
  if $Order \geq \mathcal{O}$ **then**
    ADDTORULES($Valid$)
  else
    $Distr \leftarrow D[Valid]$
    $NewOrder \leftarrow Order + 1$
    $Extended \leftarrow$ all $Source$ satisfying length($Source$)$= NewOrder$ and end with $Curr$
    if $Extended = \emptyset$ **then**
      ADDTORULES($Valid$)
    else
      for $ExtSource$ in $Extended$ **do**
        $ExtDistr \leftarrow D[Extended]$
        if KL-Divergence($Distr, ExtDistr$)$> \frac{NewOrder}{\log_2(\text{Sum}(C[ExtSource][*]))}$ **then**
          EXTENDRULE($ExtSource, ExtSource, NewOrder$)
        else
          EXTENDRULE($Valid, ExtSource, NewOrder$)

function ADDTORULES($Source$)
  if length($Source$) > 0 **then**
    $R[Source] \leftarrow C[Source]$
    $PrevSource \leftarrow$ AllButLastElement($Source$)
    ADDTORULES($PrevSource$)

algorithm S2. The network wiring algorithm.

Algorithm 2 Network wiring. Given the higher-order dependency rules R, convert the rules of variable orders into edges, perform rewiring, and output the graph G as HON.

function BUILDNETWORK(R)
    global nested dictionary $G \leftarrow \emptyset$
    $R \leftarrow$ Sort(R, ascending, by length of key $Source$)
    for r in R **do**
        G.add(r)
        if length($r.Source$)> 1 **then**
            REWIRE(r)
    REWIRETAILS()
    return G

function REWIRE(r)
    $PrevSource \leftarrow$ AllButLastElement($r.Source$)
    $PrevTarget \leftarrow$ LastElement($r.Source$)
    if (edge: (Source: $PrevSource$, Target: $r.Source$)) not in G **then**
        G.add(edge: (Source: $PrevSource$, Target: $r.Source$, Weight: $r.Probability$))
        G.remove(edge: (Source: $PrevSource$, Target: $PrevTarget$))

function REWIRETAILS()
    $ToAdd \leftarrow \emptyset; ToRemove \leftarrow \emptyset$
    for r in R **do**
        if length($r.Target$) $= 1$ **then**
            $NewTarget \leftarrow$ concatenate($Source, Target$)
            while length($NewTarget$) > 1 **do**
                if $NewTarget$ in (all Sources of R) **then**
                    $ToAdd$.add(edge:(Source: $r.Source$, Target: $NewTarget$, Weight: $r.Probability$)
                    $ToRemove$.add(edge:(Source: $r.Source$, Target: $r.Target$))
                    Break
                else
                    PopFirstElement($NewTarget$)
    $G \leftarrow (G \cup ToAdd) \backslash ToRemove$