Longitudinal analysis of multimorbidity patterns applying dynamic Bayesian networks and clustering analysis in the Andalusian Population Health Database

Supplementary Materials: methods

Contents

1.1. Dynamic Bayesian network (DBN) analysis: learning the structure

As previously described in the main document, a Dynamic Bayesian Network (DBN) is a probabilistic graphical model used to represent sequential systems. The construction of Bayesian networks typically involves two distinct steps: 1) structure learning and 2) parameter learning.

For constructing the Bayesian network structure, we performed three algorithms: the Dynamic Max-Min Hill-Climbing (DMMHC)[1] method, the particle swarm optimization algorithm for learning transition structures of higher order DBN (PSOHO)[2], and the order invariant particle swarm optimization algorithm for higher order DBNs (natPSOHO)[3].

These algorithms are available as part of the "Dynamic Bayesian Network Learning and Inference" package[4] from the R software[5]. In the following subsections, we will briefly explain these algorithms.

1.1.1. Dynamic Max-Min Hill-Climbing (DMMHC) algorithm

The DMMHC algorithm by Trabelsi et al.[1] builds 2-Time slice Bayesian networks (2T-BN). This kind of DBN satisfies the Markov property of order 1, that is, $X[t-1] \perp X[t+1] \mid X[t]$. Translated to simple words, this means that the variables in a specific time depend only on their state in the previous instant[2]. Therefore, a 2T-BN is described by a pair $(M0, M\rightarrow)$, being $M0$ the initial model and $M\rightarrow$ the transition model.

As explained by Trabelsi et al. [1], M0, or the initial model, is a BN that represents the initial joint distribution of the process $P(X[t = 0])$, consisting of a direct acyclic graph (DAG) $G0$ containing the variables $X[t = 0]$ and the set of conditional probability distributions.

M→, or transition model, is defined as another BN that represents the distribution $P(X[t+1] \mid X[t])$, consisting of a DAG $G\rightarrow$ containing the variables in $X[t] \cup X[t + 1]$

and a set of conditional distributions *P(Xi[t + 1] | paG→ (Xi))* where paG→ (Xi) are the parents of the variable *Xi[t + 1]* in *G→*, parents that can be included in time *t* or *t + 1* [1].

The DMMHC algorithm is a type of local search algorithm that is a hybrid structure learning method dealing with global model optimization and local information constraints. The DMMHC algorithm is a modification of the Max-Min Hill-Climbing algorithm by Tsamardinos et al.[6] in which the neighbourhood of a given graph is generated based on the following rules considering the hybrid method: add_edge (if the edge belongs to the set of constraints and if the result is a DAG), delete_edge and invert_edge (if the result is a DAG). The DMMHC algorithm considers local information provided by identifying neighbourhoods and symmetrical correction adapted to temporality, combined with applying a greedy search algorithm. This method will add an edge if and only if the starting node is in the neighbourhood of the ending node[1].

Trabelsi et al.[1] showed that their algorithm effectively accounts for the temporal dimension, yielding promising results. Moreover, it can be applied to high-dimensional domains (with thousands of variables). These characteristics make this algorithm a good candidate for studying the evolution of multimorbidity.

The DMMHC algorithm was extended by Quesada et al.[7], allowing the generalization of this method for Markovian orders greater than 1. This modification helps to identify connections between diseases beyond the first-order Markov property; that is, variables in a specific time could depend not only on their state in the previous instant. They applied the following formula to calculate the joint probability given an arbitrary Markovian order:

$$p(X^{0:T}) = p(X^0) \prod_{t=0}^{m-1} p(X^{t+1}|X^{0:t}) \prod_{t=m}^{T-1} p(X^{t+1}|X^{(t-m+1):t}) \quad (1)$$

where X^t are all the nodes of time slice *t*, *m* is the chosen Markovian order, *0* is the oldest time slice and *T* the most recent. Quesada implemented this modification in the "Dynamic Bayesian Network Learning and Inference" R package[4].

Another modification implemented in the "Dynamic Bayesian Network Learning and Inference" R package[4] consisted in how it measures and filters the condition independence given the data. It uses the exact t-test for Pearson's correlation coefficient for this purpose, applying the following formula[8]:

$$t(X, Y|\boldsymbol{Z}) = \rho_{X,Y|\mathbf{Z}} \sqrt{\frac{n - |\mathbf{Z}| - 2}{1 - \rho^2_{X,Y|\mathbf{Z}}}} \quad (2)$$

where $\rho_{X,Y|\mathbf{Z}}$ is the partial correlation coefficient of *X* and *Y* given the set of variables **Z**; and |**Z**| is the number of nodes in the set **Z**.

In summary, the DMMHC algorithm first performs a local search of the structure around each node. This step is performed by finding the Markov blanket of each node, which is the group of nodes that makes it conditionally independent from the other nodes[8]. To measure the conditional independence, the algorithm applies the t-test for Pearson's correlation coefficient to find the set of nodes $\boldsymbol{Z}$ that make a node *X* independent from all the nodes $Y \in \boldsymbol{Y}$. Then the local structures of all the nodes are combined, constraining the step of scoring and finding the best general structure. The Bayesian information criterion (BIC)[9] is used to score the resulting structures and select the model.

1.1.2. The Particle Swarm Optimization algorithm for learning transition structures of higher-order DBN (HO-DBN or PSOHO algorithm)

The PSOHO algorithm, developed by Santos et al.[2], is another option considered in this analysis. The assumption of first-order Markov that frequently use DBNs could be too

restrictive for some cases, particularly, when there are different time delays[2]. An alternative could be the use of HO-DBN, defined by relaxing the first-order Markov property of DBN to higher orders.

However, this approach also has certain limitations due to the high number of nodes and the associated computational cost. To address these challenges, the algorithm has developed a technique that applies Particle Swarm Optimization for learning the transition models.

As described by Santos et al. [2], this technique defines a population of particles moving in the search space following (a) their best position found (i.e., local best), (b) the best position found in all neighbours (i.e., global best) and (c) a random perturbation factor in trajectory. This population is analyzed at each time step, calculating each particle's position score and each particle updates their velocity and position according to the following formula:

$$\mathrm{v_i}(t\ +\ 1) = \omega \mathrm{v_i}(t)\ +\ \psi_1 \mathrm{R_1}(\mathrm{x_{gi}} -\ \mathrm{x_i}(t)) +\ \psi_2\ \mathrm{R_2}(\mathrm{x_{bi}} - \mathrm{x_i}(t)) \quad (3)$$

$$x_i(t+1) =\ x_i(t) +\ v_i(t+1) \quad (4)$$

v_i and x_i are the velocity and position of a particle i; R_1 and R_2 are random numbers in the interval [0, 1] — a perturbation factor used in the formula to explore the search space; x_{gi} is the global best of the particle, and x_{bi} local best of the particle. Finally, ψ_1 and ψ_2 are the social and cognitive parameters, and ω is an inertia factor. ψ_1, ψ_2 and ω are given a priori and control the behavior of the algorithm. The algorithm iterates until a criterion is met (i.e., number of iterations, amount of time without improving the global best position). To better understand this algorithm, it is important to define position, velocities and inertia in relation to DBN following the paper by Santos et al. [2]:

1. Position. A causal unit *CUXi(t-j)* is defined as a list containing each variable that is a parent of a given variable X_i in the previous time $t - j$. A casual list $CL^{(t-j)}$ in time (t-j) as a list of causal units for each variable in the model. Finally, the HO-DBN transition structure is represented as a set of causality lists, one for each previous time step defined by the Markovian order.

2. Velocity. It is defined as a set of causality lists with the same number of variables and order in which values in the causal units may have negative or positive signals. There are set of signed causality lists, with signed causal units. Therefore, $-X_i$ in a signed causal unit of a variable X_j at time *(t-h)* represents the deletion of the edge, and $+X_i$ represents an addition.

3. Inertia. This concept is defined by Santos et al.[2] to introduce extra randomness in the algorithm. They proposed that each particle should not keep its current velocity at each time step but adopt a new random velocity.

According to Santos et al.[2] the PSOHO algorithm outperforms greedy algorithms in convergence time except in cases with few variables. This difference is even higher as the number of variables or Markovian order grows. All these characteristics make the PSOHO algorithm a strong candidate to be applied in this research project.

Based on the documentation from the "Dynamic Bayesian Network Learning and Inference" R package[4] no additions were performed to the PSOHO algorithm.

1.1.3. Order invariant particle swarm optimization algorithm for higher-order DBNs (natPSOHO)

The natPSOHO algorithm, developed by Quesada et al.[3], proposed a structure learning method very similar to the PSOHO method by Santos et al.[2]. They used a particle swarm

optimization to search in the space of possible structures and added an order-invariant encoding to avoid additional costs of increasing the Markovian order. This encoding represents the network as vectors of natural numbers to set a maximum desired order. According to Quesada et al.[3], their method exhibits efficiency in high orders, outperforming similar methods in both execution time and quality of the obtained networks. Therefore, the natPSOHO algorithm was another good candidate for this research project.

Based on the documentation from the "Dynamic Bayesian Network Learning and Inference" R package[4] no modifications were performed to the natPSOHO algorithm.

1.2. Dynamic Bayesian network (DBN) analysis: learning the parameters

After learning the structure, the parameters are fitted via maximum likelihood estimation for continuous data. The Bayesian approach was performed for parameter learning, using the "Dynamic Bayesian Network Learning and Inference"[4] and "Bayesian Network Structure Learning, Parameter Learning and Inference" [10] R[5] packages. Initially, this project focused on learning the structure of the network. Learning the parameters is key step in building the predictive model, which will be addressed in future work.

1.3. Clustering analysis: Community detection algorithms

Community detection algorithms aim to find the community structure of the network analyzed. We used modularity to search for clusters of diseases by applying the Louvain and Leiden algorithms[11]. As explained in the main document, modularity for weighted networks is an objective function defined as the fraction of weighted edges falling within clusters minus the expected fraction of weighted edges falling within clusters[12]. Its formula for weighted networks is as follows[13]:

$$Modularity\ for\ weighted\ networks = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j) \quad (5)$$

Here, A_{ij} represents the weight of the edge (i, j); c_i is the cluster to which node i is assigned; δ-function $\delta(u, v)$ is an expression which is equal to 1 when the nodes u and v belong to the same cluster and 0 otherwise; $k_i = \sum_j A_{ij}$ corresponds to the sum of the weights of the edges connected to node i; and $m = \frac{1}{2}\sum_{i,j} A_{ij}$

The following subsections will briefly explain the Louvain and Leiden algorithms and how they apply modularity as their objective function.

1.3.1. Louvain algorithm

The initialization of this method consists of a clustering where all nodes (i.e., diseases in our study) are in a separate cluster. This method applies two iteratively repeated steps. The first one computes the change of the modularity for every node due to the transfer of the node to another adjacent cluster. An adjacent cluster of disease i is defined as any community with disease j adjacent to i. All nodes are processed sequentially. When the modularity is increased, the transfer is done until no improvement is achieved. In this step, there is a modularity optimization. Subsequently, in the second step, the clusters obtained in the first step are aggregated into condensed nodes and new edges are created connecting these condensed nodes. The weight of the new edges is the sum of the weights of the edges between the clusters in the first step. Then, a new iteration occurs, and step one starts again with the new condensed nodes and edges. The iteration continues until there is no change in the first step; that is, there are no transfers of nodes that increase modularity and, therefore, no improvement in the modularity optimization.

For more comprehensive information regarding the Louvain algorithm, we recommend referring to the paper authored by Blondel *et al.*[13].

1.3.2. Leiden algorithm

This method also begins with a singleton partition and iteratively follows three steps. The first step is the local moving of nodes to optimize modularity, obtaining a partition *P*, like the Louvain method. The second step is an addition to the Louvain algorithm and consists of refining the partition obtained previously, $P \rightarrow P_{refined}$. In the third step, there is also a community aggregation step, like in the Louvain algorithm.

In the refinement phase, clusters in *P* may be split into subcommunities in $P_{refined}$, but not always. This phase starts with a singleton partition $P_{refined}$; that is, each node is a community. Then, there is a local moving of nodes in $P_{refined}$. In this step, the transfer of edges and merge of communities can only be done within each community of the partition *P*, except for nodes that are on their own in $P_{refined}$ that can be merged with another community. In all cases, a node (i.e., disease in our case) can be merged with another community in $P_{refined}$ only if both are well connected to their community in *P*. Leiden algorithm guarantees well-connected communities compared to the Louvain method[14], which is an important difference to consider.

At the same time, the selection of the community is a random process, although the quality function must increase. These characteristics increase the efficiency of the refinement phase, exploring the partition space more broadly.

It is also important to consider that the initial partition used in the community aggregation phase is based on the partition *P*, like in the Louvain algorithm. This step considers different subcommunities (obtained in $P_{refined}$) as part of the same community (obtained in *P*).

For more comprehensive information regarding the Leiden algorithm, we recommend referring to the paper authored by Traag *et al.*[14].

1.3.3 Clustering analysis: comparison

As explained in the main document, to compare both algorithms, we considered modularity (quality), the number of clusters, and the resolution parameter. The higher the modularity, the better the quality of the partition obtained. Regarding the resolution parameter, higher resolution leads to more groups, working as a threshold. This parameter is part of the modularity formula in the following way[14]:

$$Modularity = \frac{1}{2m}\sum_{c}(e_c - \gamma\frac{K_c^2}{2m}) \quad (6)$$

Here, e_c is the actual number of edges in cluster c; γ is the resolution parameter; K_c is the sum of the 'nodes' degrees in cluster c; m is the 'network's total number of edges; $K_c^2/2m$ is the expression of the expected number of edges.

1.4. Centrality analysis: Network topology score formulas

Three scores were calculated: degree, closeness centrality, and harmonic centrality.

The degree is defined as the number of edges a node has. We also calculated the weighted degree as the sum of the weighted edges.

Closeness centrality measures the mean distance of a node to the other nodes in the network.

$$\text{Closeness centrality} = \frac{\mathrm{n}-1}{\mathrm{n}-1}\,\frac{\mathrm{n}-1}{\sum_{\mathrm{v}=1}^{\mathrm{n}-1}\mathrm{d}(\mathrm{v},\mathrm{u})} \quad (7)$$

Here, *n-1* is the number of nodes that u can reach; *d(v, u)* is the distance between *u* and v; and *N* is the total number of nodes in the network.

Harmonic centrality is similar to closeness centrality but uses harmonic mean instead of arithmetic mean[15].

$$Harmonic\ centrality = \frac{1}{N-1}\sum_{v \neq u}\frac{1}{d(v,u)} \quad (8)$$

References

1. Trabelsi, G., Leray, P., Ben Ayed, M. & Alimi, A. M. Dynamic MMHC: A Local Search Algorithm for Dynamic Bayesian Network Structure Learning BT - Advances in Intelligent Data Analysis XII. in (eds. Tucker, A., Höppner, F., Siebes, A. & Swift, S.) 392–403 (Springer Berlin Heidelberg, 2013).
2. Santos, F. P. & Maciel, C. D. A PSO approach for learning transition structures of Higher-Order Dynamic Bayesian Networks. in *5th ISSNIP-IEEE Biosignals and Biorobotics Conference (2014): Biosignals and Robotics for Better and Safer Living (BRC)* 1–6 (2014). doi:10.1109/BRC.2014.6880957
3. Quesada, D., Bielza, C. & Larrañaga, P. Structure Learning of High-Order Dynamic Bayesian Networks via Particle Swarm Optimization with Order Invariant Encoding BT - Hybrid Artificial Intelligent Systems. in (eds. Sanjurjo González, H., Pastor López, I., García Bringas, P., Quintián, H. & Corchado, E.) 158–171 (Springer International Publishing, 2021).
4. Quesada, D. & Valverde, G. Dynamic Bayesian Network Learning and Inference. https://cran.r-project.org/web/packages/dbnR/dbnR. (2022).
5. R Core Team. R: A language and environment for statistical computing. https://www.r-project.org/ (2022).
6. Tsamardinos, I., Brown, L. E. & Aliferis, C. F. The max-min hill-climbing Bayesian network structure learning algorithm. *Mach. Learn.* **65**, 31–78 (2006).
7. Quesada, D., Bielza, C., Fontán, P. & Larrañaga, P. Piecewise forecasting of nonlinear time series with model tree dynamic Bayesian networks. *Int. J. Intell. Syst.* **37**, 9108–9137 (2022).
8. Quesada, D., Valverde, G., Larrañaga, P. & Bielza, C. Long-term forecasting of multivariate time series in industrial furnaces with dynamic Gaussian Bayesian

networks. *Eng. Appl. Artif. Intell.* **103**, 104301 (2021).

9. Schwarz, G. Estimating the dimension of a model. *Ann. Stat.* **6**, 461–464 (1978).
10. Marco, S., Tomi, S. & Robert, N. Bayesian Network Structure Learning, Parameter Learning and Inference. https://www.bnlearn.com/documentation/bnlearn-manu (2023).
11. Lancichinetti, A. & Fortunato, S. Community detection algorithms: A comparative analysis. *Phys. Rev. E* **80**, 056117 (2009).
12. S, H. Performance evaluation of community detection algorithms in multivariate clustered binary. (Eindhoven University of Technology, 2022).
13. Blondel, V. D., Guillaume, J.-L., Lambiotte, R. & Lefebvre, E. Fast unfolding of communities in large networks. *J. Stat. Mech. Theory Exp.* **2008**, P10008 (2008).
14. Traag, V. A., Waltman, L. & van Eck, N. J. From Louvain to Leiden: guaranteeing well-connected communities. *Sci. Rep.* **9**, 1–12 (2019).
15. St Luce, S. & Sayama, H. Analysis and Visualization of High-Dimensional Dynamical Systems' Phase Space Using a Network-Based Approach. *Complexity* **2022**, 1–11 (2022).