Model for efficient dynamical ranking in networks

Andrea Della Vecchia [1,2,*,†] Kibidi Neocosmos [3,4,5,*,‡] Daniel B. Larremore [6,7,§]
Cristopher Moore [8,‖] and Caterina De Bacco [3,¶]

[1] *Istituto Italiano di Tecnologia, 16163 Genoa, Italy*
[2] *MaLGa Center, Università di Genova, 16146 Genoa, Italy*
[3] *Max Planck Institute for Intelligent Systems, 72076 Tübingen, Germany*
[4] *African Institute for Mathematical Sciences, Cape Town 7950, South Africa*
[5] *University of Tübingen, 72074 Tübingen, Germany*
[6] *Department of Computer Science, University of Colorado Boulder, Boulder, Colorado 80309, USA*
[7] *BioFrontiers Institute, University of Colorado Boulder, Boulder, Colorado 80309, USA*
[8] *Santa Fe Institute, Santa Fe, New Mexico 87501, USA*

(Received 31 July 2023; accepted 2 August 2024; published 12 September 2024)

We present a physics-inspired method for inferring dynamic rankings in directed temporal networks—networks in which each directed and timestamped edge reflects the outcome and timing of a pairwise interaction. The inferred ranking of each node is real-valued and varies in time as each new edge, encoding an outcome like a win or loss, raises or lowers the node's estimated strength or prestige, as is often observed in real scenarios including sequences of games, tournaments, or interactions in animal hierarchies. Our method works by solving a linear system of equations and requires only one parameter to be tuned. As a result, the corresponding algorithm is scalable and efficient. We test our method by evaluating its ability to predict interactions (edges' existence) and their outcomes (edges' directions) in a variety of applications, including both synthetic and real data. Our analysis shows that in many cases our method's performance is better than existing methods for predicting dynamic rankings and interaction outcomes.

DOI: 10.1103/PhysRevE.110.034310

I. INTRODUCTION

When considering a collection of people, animals, teams, or other entities, there is often an underlying hierarchy structuring the system. This hierarchy may be formally instilled in the sense that some individuals are explicitly granted certain ranks based on positions of authority. For example, in a school, there are students, teachers, and the principal or head of the school, with each position explicitly known and ranked in terms of level of authority. Alternatively, a hierarchy may be implicit in the sense that the ranks are not explicitly granted or known, but instead encoded in behaviors or interactions. For example, in animal dominance hierarchies, animals may be preferentially aggressive toward those lower in rank. In both explicit and implicit cases, hierarchies can be determined by analyzing the patterns of interactions between the entities of the system.

If we wish to infer the ranks of entities in a hierarchical structure from the patterns of their interactions, then we can treat ranks as either static or dynamic and as ordinal or real-valued.

In the static case, time is irrelevant, and we treat all the interactions at once regardless of the sequence in which they occur, as one might when ranking the teams in a sports league at the end of a seasons.

In the dynamic case, each individual's ranking may rise or fall over time, retaining the memory of past interactions while taking new interactions into account. This can be seen in leagues such as the U.S. National Basketball Association (NBA) where rankings derived from recent games provide insight for predicting games in the near future, yet the rankings themselves may nevertheless change slowly over the course of a season or seasons. We are also interested in real-valued ranks, rather than ordinal ranks, such that the size of rank difference between two entities is an interpretable and predictive quantity, regardless of whether they are adjacent or well separated in ordinal rank.

To model systems of this type we propose Dynamical SpringRank. This builds on the previously proposed SpringRank algorithm [1] by incorporating time information, inferring a dynamic hierarchy from a dynamic network: that is, a dataset of timestamped interactions, each of which defines a directed edge $i \to j$ indicating that i "beat" j at time t. We make similar physically-inspired assumptions as

*These authors contributed equally to this work.
†Contact author: andrea.dellavecchia@iit.it
‡Contact author: kibidi.neocosmos@tuebingen.mpg.de
§Contact author: daniel.larremore@colorado.edu
‖Contact author: moore@santafe.edu
¶Contact author: caterina.debacco@tuebingen.mpg.de

SpringRank, modeling directed edges as springs and assuming that entities are more likely to interact if their ranks are not too far apart. We also propose a generative model for constructing directed, hierarchical networks that evolve over time.

Finally, we evaluate Dynamical SpringRank on a variety of synthetic and real datasets. From our findings, we conclude that it accurately and efficiently infers ranks and predicts the direction of edges in dynamic settings. Furthermore, it frequently outperforms other algorithms such as the Elo Rating System and Whole-History Rating.

II. RELATED WORK

Estimating hidden hierarchies from pairwise interactions is a fundamental problem in a wide variety of contexts. Several models have been proposed to study *static* hierarchies, i.e., scenarios where ranks do not change in time. One line of research considers spectral methods, which exploit eigenvalues and eigenvectors of certain matrices that can be built from the network structure given an input. These methods learn real-valued scores on nodes and differ in the choice of the underlying matrix considered to solve an eigenvalue problem. Prominent examples include Eigenvector Centrality [2], PageRank [3], and Rank Centrality [4]. A different line of research considers ordinal rankings, where nodes are assigned an order rather than a real-valued score. Examples are Minimum Violation Rank [5–7], Ranked Stochastic Block Model [8], SerialRank [9] and SyncRank [10]. Another main line of research is that of probabilistic approaches, where a main assumption is that outcomes are random variables, and they depend on real-valued scores. These are learned using techniques from statistical inference and can be used to estimate the probability of an outcome. These approaches are considered in various domains. For instance, in economics and psychology, Random Utility Models [11] investigate preferences for choices that are not deterministic. A relevant example is the Bradley-Terry-Luce (BTL) model [12,13]. In ecology, probabilistic niche models [14–16] are used to study food webs. In social networks, a variety of probabilistic approaches have been considered. They differ in their assumptions about the underlying patterns playing a role in determining the hierarchy. For instance, social status can be considered to model friendship [17]. A combination of hierarchy and community structure [18] can be used to learn directed interactions between individuals. Latent space models assume that each node has a position in an underlying latent space [19]. Physics-inspired models draw from analogies with physical systems, for instance a system of springs as in SpringRank [1] or continuous spin systems [20].

In contrast, *dynamic* approaches model dynamic environments where ranks vary in time and interactions have a relevant chronological order. For instance, the Elo Rating System [21], commonly used for rating chess players, is one of the most popular online methods. It was later improved by the Glicko system [22], which incorporates a measure of reliability in estimating ranks to capture their uncertainty due to, for instance, a period of inactivity or lack of data. The Dynamic TranSync model [23] assumes that observations are noisy measurements of strength differences with zero-mean noise and imposes smoothness constraints on the time-varying strengths. Another approach is a win-loss ranking algorithm [24] and its dynamic extension [25]. A Bayesian ranking system inferring individual ranks from team-level outcomes is the so called TrueSkill algorithm [26], which can be seen as a generalization of the Elo system. This has been extended by TrueSkill Through Time (TTT) [27] which infers smooth time series of ranks. Decaying-history ratings such as [25] act directly on the data observations, progressively forgetting old interactions. One drawback of this approach is that time decay increases the uncertainty of player ratings: players who stop playing for a while may experience huge jumps in their ratings when they start playing again. However, players who play very frequently may have the feeling that their rating is stuck. If players do not all play at the same frequency, then there is no clear way to tune the decay rate [28].

Finally, an additional type of dynamic method treats ranks as time-varying, but infers the ranks at each time-step by considering the totality of all observations, including those before and after any particular time step. For instance, the Whole-History Rating (WHR) [28], a Bayesian approach based on the dynamic Bradley-Terry-Luce model, computes the exact maximum a posteriori estimate of ranks over the whole history of all players.

III. THE MODEL

We represent a series of interactions between N individuals as a sequence of weighted directed networks with adjacency matrix A^t for $t = 0, 1, 2, \ldots, T$. For each t, its entry A^t_{ij} is the outcome of interactions $i \to j$ suggesting that i is ranked above j. This allows both cardinal and ordinal inputs. For instance, in team sports, A^t_{ij} could be the number of points by which team i beat team j, or we could simply set $A^t_{ij} = 1$ to indicate that i won and j lost. We can include the case where individuals interact multiple times at time t by summing the corresponding entries.

We assume that the values of A^t_{ij} are influenced by a vector of real-valued ranks $\mathbf{s}^t = (s^t_1, \ldots, s^t_N)$, where s^t_i is i's strength or prestige at time t. To model these interactions, we follow SpringRank's approach of imagining the network as a physical system [1]. Specifically, each node i is embedded in $\mathbb{R}$ at position s^t_i, and each directed edge $i \to j$ becomes an oriented spring with a nonzero resting length and displacement $s^t_i - s^t_j$. Since we are free to rescale latent space and the energy scale, we set the spring constant and resting length to 1. The spring corresponding to an edge $i \to j$ at time t then has energy

$$H_{ij}\left(s^t_i, s^t_j\right) = \tfrac{1}{2}\left(s^t_i - s^t_j - 1\right)^2. \tag{1}$$

If there were no other effects, then the total energy of the system at time t would be

$$H^t(\mathbf{s}^t) = \sum_{i,j=1}^{N} A^t_{ij} H_{ij}\left(s^t_i, s^t_j\right). \tag{2}$$

If we determined $\mathbf{s}^t$ by minimizing H^t for each t separately, then we would simply be applying the static SpringRank model separately to each "snapshot" of the network. This would ignore all previous (and future) interactions, and ignore the hypothesis that ranks change smoothly from one time-step to the next.

To model this smoothness, we also assume a dependence between ranks at successive time-steps. Specifically, we extend the Hamiltonian (2) with an extra term that models the *self-interaction* between past and current ranks,

$$H^t_{\text{self}}(\mathbf{s}^t, \mathbf{s}^{t-1}) = \frac{k}{2}\sum_{i=1}^{N}\left(s_i^t - s_i^{t-1}\right)^2. \tag{3}$$

This can be seen as a set of additional "self-springs" that connect the rank of each individual with its own previous rank. The spring constant k parametrizes how smoothly we want the ranks to change from one step to the next. In inference terms, k is a hyperparameter which we tune using cross-validation.

Summing over all time-steps $0 < t \leqslant T$ and adding this to the pairwise interactions at each time-step then gives a total energy

$$H_{\text{total}}(\{\mathbf{s}^t\}) = \sum_{t=0}^{T} H^t(\mathbf{s}^t) + \sum_{t=1}^{T} H^t_{\text{self}}(\mathbf{s}^t, \mathbf{s}^{t-1}). \tag{4}$$

We call this the dynamical SpringRank Hamiltonian. The optimal ranks $\mathbf{s}^0, \mathbf{s}^1, \ldots, \mathbf{s}^T$ are those that minimize it.

There are two ways to minimize H_{total}. One is to proceed in an online way, moving forward in time. In this approach, we use the static SpringRank model Eq. (2) to find the initial ranks $\mathbf{s}^0$ by minimizing $H^0(\mathbf{s}^0)$. As in Ref. [1], the energy is unchanged if we add a constant to all the ranks; we can break this translational symmetry by setting the mean initial rank $(1/N)\sum_{i=1}^{N} s_i^0$ to zero. Then, at each subsequent time-step $t \geqslant 1$, we update the ranks by taking into account both the new pairwise interactions and the self-springs connecting the ranks with their previous values. Namely, given $\mathbf{s}^{t-1}$ and A^t, we find the ranks $\mathbf{s}^t$ that minimize $H^t(\mathbf{s}^t) + H^t_{\text{self}}(\mathbf{s}^t, \mathbf{s}^{t-1})$.

Since this is a convex function of $\mathbf{s}^t$, we can find its minimum by setting its gradient to zero, or equivalently by balancing all the forces s_i^t. This yields a system of linear equations:

$$\begin{aligned}&[D^{\text{out},t} + D^{\text{in},t} - (A^t + (A^t)^\dagger) + k\mathbb{I}]\,\mathbf{s}^t \\ &\quad = [D^{\text{out},t} - D^{\text{in},t}]\mathbf{1} + k\mathbf{s}^{t-1}.\end{aligned} \tag{5}$$

Here $D^{\text{out},t}$ and $D^{\text{in},t}$ are diagonal matrices whose entries are the weighted out- and in-degrees $D^{\text{out},t}_{ii} = \sum_j A^t_{ij}$ and $D^{\text{in},t}_{ii} = \sum_j A^t_{ji}$; $\dagger$ denotes the transpose; $\mathbb{I}$ is the identity matrix; and $\mathbf{1}$ is the all-ones vector. The derivation of Eq. (5) can be found in Appendix A.

The matrix on the left-hand side (LHS) of Eq. (5) is invertible for $k > 0$. This can be proved following the same reasoning as in Ref. [1] under Eq. (3) and noticing that the LHS of our Eq. (5) coincides with the LHS of Eq. (5) in Ref. [1] when replacing k with α. Thus, for each A^t and each $\mathbf{s}^{t-1}$, Eq. (5) has a unique solution $\mathbf{s}^t$. Overall, Eq. (5) is similar to the regularized version of SpringRank [1] with regularization parameter $\alpha = k$. However, unlike the static model, there is a term on the right-hand side containing the previous ranks $\mathbf{s}^{t-1}$, creating a Markovian dependence between successive time-steps. We refer to this model as Dynamical SpringRank (abbreviated as DSR). We provide a visual representation of the model in Fig. 1.

Importantly the online DSR approach does not actually minimize H_{total}, instead solving a sequence of minimization problems, one for each time step. To minimize H_{total} instead,

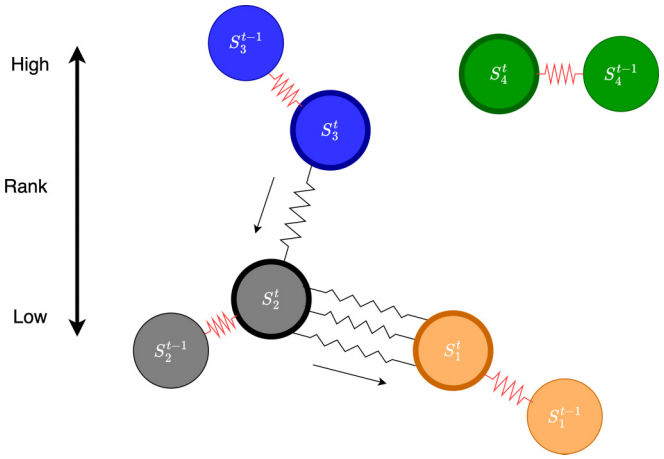

FIG. 1. A visual representation of Dynamical SpringRank. Each node i has rank s_i at time t and each edge is represented as a spring. The red springs indicate self-springs that connect past and present ranks. The black springs indicate interactions with different entities. The blue and gray nodes interact once while the gray and gold nodes interact three times. In contrast, the green node does not interact with the other entities. Arrows indicate the direction of a win in a directed interaction between two nodes.

we set $\nabla H_{\text{total}}(\mathbf{s}^t) = 0$, solving for the minimizers $\mathbf{s}^t$ over all $N(T+1)$ ranks simultaneously, yielding the following system of equations (Appendix B):

$$\begin{aligned}&[D^{\text{out},t} + D^{\text{in},t} - (A^t + (A^t)^\dagger) + 2k\mathbb{I}]\,\mathbf{s}^t \\ &\quad = [D^{\text{out},t} - D^{\text{in},t}]\mathbf{1} + k(\mathbf{s}^{t-1} + \mathbf{s}^{t+1}).\end{aligned} \tag{6}$$

This differs from Eq. (4) in that the right-hand side now includes both past and future ranks (which doubles the contribution of k on the left). We remove the terms $\mathbf{s}^{t-1}$ and $\mathbf{s}^{t+1}$ for $t = 0$ and $t = T$, respectively. This is equivalent to specifying boundary conditions $\mathbf{s}^{-1} = \mathbf{s}^{T+1} = 0$, i.e., ranks outside the considered time interval are set to zero. Other possible choices could be made for these boundary conditions. They mainly impact the values of ranks close to the boundaries and their effect lessens in the presence of many time steps. See Appendix G for further discussion.

This entire system has translational symmetry, since the energy Eq. (4) remains the same if we add the same constant to all ranks at all times, but we can again break this symmetry by setting the mean rank to zero.

Additionally, in contrast to Eq. (5), the ranks at t now depend on both $t-1$ and $t+1$, which themselves depend on ranks at adjacent time-steps, so that ranks are affected by interactions in both the past and the future. In computer science, methods like this where the entire history is provided to the algorithm are called *offline*, to distinguish them from *online* approaches that update their results in real time as data becomes available. Thus, we refer to this model as Offline Dynamical SpringRank (OFFDSR).

The cost of solving Eq. (5) for a single time-step is the same as static SpringRank with only one additional parameter to be tuned using cross-validation, and there are T such N-dimensional equations to be solved successively. However, Eq. (6) requires solving a single system of dimension NT, whose operator consists of T blocks, each of dimension

$N \times N$. While these two approaches feature numbers of nonzero entries that are fundamentally determined by the number of total edges across all time steps, the cost of solving DSR vs OFFDSR will depend on the particular choice of linear solver [29].

Philosophically, Eqs. (5) and (6) are trying to do two different things. If we are given all the data $A^0, A^1, \ldots, A^T$ and we want to infer retrospectively how each individual's rank changed over time, then it makes sense to include both past and future interactions as in Eq. (6) so that s_i^t is affected by i's entire history. In contrast, Eq. (5) can be viewed as modeling each individual's perceived rank s_i^t at a time $t \leqslant T$ in the past, based only on the interactions that have occurred so far, thus ignoring the future steps $t+1, \ldots, T$.

In principle, one could envisage other ways to formally incorporate an explicit dependence on $\mathbf{s}^{t-1}$ into the model, and we provide one example in Appendix C. However, we found that the approaches presented in this section provide a natural interpretation, result in good prediction performance on both real and synthetic datasets (see Sec. IV) and are computationally scalable.

We close this section with two possible extensions to these models. First, in some settings we might have timestamps t that are not successive integers $0, 1, \ldots, T$. In this case, if the time interval between two successive times is Δt, then one could scale the spring constant of the self-springs between time-steps as $k/\Delta t$. This corresponds to the fact that if we have Δ identical springs in series, each of which is stretched by $(s^t - s^{t-1})/\Delta$, their total energy is $(1/2)(k/\Delta)(s^t - s^{t-1})^2$. The same expression applies if the timestamps are real-valued so that Δ is not an integer.

Second, if we believe that not just the ranks themselves but their rates of change behave smoothly over time, then one could add a momentum term to the Hamiltonian which is quadratic in the discrete second derivative of the ranks. Since

$$\begin{aligned}&((s^{t+1} - s^t) - (s^t - s^{t-1}))^2\\ &\quad = (s^{t+1} - 2s^t + s^{t-1})^2\\ &\quad = 2(s^t - s^{t-1})^2 + 2(s^{t+1} - s^t)^2 - (s^{t+1} - s^{t-1})^2,\end{aligned}$$

this is equivalent to adding a repulsive force, i.e., a spring with negative spring constant, between ranks two time-steps apart. Note that the system nevertheless remains convex: this momentum term is positive semidefinite, so adding it to Eq. (4) keeps the coupling matrix positive definite except for translational symmetry. Of course, these terms are second-order in time. In the online approach, one would have to determine $\mathbf{s}^0$ from the static model, $\mathbf{s}^1$ from the first-order model (5), and then use the model including this momentum term for $\mathbf{s}^t$ for $t \geqslant 2$. We have not pursued this here, but it may make sense for certain datasets.

A. Moving-window SpringRank

Before we test the various versions of Dynamical SpringRank defined above, we consider a simpler model as a baseline. The simplest way to extend SpringRank to a dynamical context is to apply the static model to the interactions in a series of "windows," where in each window we sum the interactions over a series of consecutive time-steps. For instance, we can compute $\mathbf{s}^t$ for each t by applying the static model to a window of width τ, i.e., replacing A^t with $\sum_{t'=t}^{t+\tau-1} A^{t'}$. Since these windows overlap, the resulting estimates $\mathbf{s}^t$ will be smooth to some extent, even without imposing an explicit dependence between $\mathbf{s}^t$ and $\mathbf{s}^{t-1}$. We use this method, which we call moving-window SpringRank (mwSR), as a baseline to compare with the dynamical models presented above.

Roughly speaking, a larger τ is like a larger self-spring constant k, since it induces more overlap between windows and thus a stronger correlation between the inferred ranks. However, like a decaying-history approach, mwSR assumes a particular kernel for the importance of past time-steps: namely, that all t' in the window are equally important. In contrast, Dynamical SpringRank infers the importance of past time-steps by coupling $\mathbf{s}^t$ with $\mathbf{s}^{t-1}$.

However, both models have a free parameter that needs to be tuned, i.e., k and τ. A shorter window τ or smaller spring constant k allows the ranks to respond quickly to new interactions, while a longer window or larger spring constant more tightly couples nearby estimates. This trade-off suggests the existence of an optimal window length τ_{opt}. We tune τ using a cross-validation procedure as explained in Appendix F.

B. Generative model and synthetic data

Analogous to a model presented in Ref. [1], we propose a probabilistic generative model for dynamic data. It takes as input the ranks $\mathbf{s}^t$ and generates a sequence of weighted directed networks with adjacency matrix A^t at time t. One can also imagine models that generate the ranks, for instance with a random walk with Gaussian steps whose log-probability is the self-spring Hamiltonian (3), but we treat $\mathbf{s}^t$ as an input since we want the user of this model to have control over how the ground-truth ranks vary with time. For instance, in our experiments below we generate synthetic data where the ranks vary sinusoidally.

The generative model has two real-valued parameters: a signal-to-noise ratio or inverse temperature β, and an overall density of edges c. Given the ranks $\mathbf{s}^t$, it generates weighted, directed edges between each pair of nodes i, j independently, as follows. The probability $P_{ij}^t(\beta)$ of i "beating" j at time t, giving a directed edge $i \to j$, is a logistic function as in Ref. [1] or the Bradley-Terry-Luce model [12,13]:

$$P_{ij}^t(\beta) = \frac{1}{1 + \mathrm{e}^{-2\beta(s_i^t - s_j^t)}}.$$

The number of such edges, which gives the integer weight A_{ij}^t, is then drawn from a Poisson distribution whose mean λ_{ij}^t is $cP_{ij}^t(\beta)$:

$$A_{ij}^t \sim \mathrm{Poi}\left(\lambda_{ij}^t = \frac{c}{1 + \mathrm{e}^{-2\beta(s_i^t - s_j^t)}}\right). \tag{7}$$

Since $P_{ij}^t(\beta) + P_{ji}^t(\beta) = 1$, for any pair i, j the total number of interactions $A_{ij}^t + A_{ji}^t$ is Poisson-distributed with mean c. The rank differences $s_i^t - s_j^t$ are used only to choose the directions of these edges. This is equivalent to a model where we define a random multigraph where the number of edges between i and j is $\mathrm{Poi}(c)$, and then we choose the direction of each edge independently according to P_{ij}^t.

This is different from the generative model proposed in the static case in Ref. [1]. In that model the probability that i and j interact depends on $s_i - s_j$ so that nodes are more likely to interact if their ranks are fairly close. This is consistent with SpringRank's assumption that if i beats j then j is below i, but not too far below it (since the springs have resting length 1). This assumption makes sense for some datasets but not for others. By generating synthetic data without this dependence, our intent is to pose a greater challenge to SpringRank by modeling (for example) round-robin tournaments where every team plays each other.

C. Model evaluation

Assessing a ranking model on real datasets is not straightforward since we do not know the true values of the underlying ranks. Nevertheless, we may measure the extent to which inferred ranks are accurate in the sense that they can predict the outcome of new observations.

There are several performance metrics that can be used for prediction evaluation. From coarse-grained measures capable of predicting the likely winner to more fine-grained measures that also estimate odds, we consider four main metrics in our experiments, detailed in Appendix D. We measure prediction performance using a cross-validation protocol where datasets are divided into training and test sets. The training set is used for hyperparameter tuning and parameter estimation while performance is evaluated on the test set. To preserve the chronological ordering of the data, the test set contains future observations, i.e., observations that chronologically follow those used in training. Hyperparameters for each method are tuned using grid-search to maximize the performance metrics as described in Appendix F.

IV. RESULTS

Having introduced Dynamical SpringRank and its generative counterpart, as well as discussing model selection between the dynamic and static versions of SpringRank, we now illustrate their behavior on synthetic and real data.

We compare prediction performance on held-out test data for DSR and OFFDSR against several state-of-the-art algorithms such as the Elo Rating System (Elo) [21], TrueSkill (TS) [27], "win-loss" decay-history rating (W-L) [25], and Whole-History Rating (WHR) [28] (see Appendix E for a brief description of these methods). In addition, we consider two baselines: static SpringRank (SR) [1] and mwSR presented above in Sec. III A. (Note that static SpringRank is the limiting case of mwSR with one window covering the entire dataset.) Additionally, since OFFDSR considers future information, in the experiments it was only given past information so that a fair comparison can be made with the other models in terms of prediction performance (see Appendix G for a further discussion).

A. Performance on synthetic data

We first consider synthetic data, generated as described in Sec. III B, in which ranks evolve according to periodic ground truth dynamics,

$$s_i^t = b_i \cos(\omega_i t + \phi_i) + c_i \cos(\upsilon_i t + \phi_i), \tag{8}$$

where b_i, c_i, ω_i, ϕ_i, υ_i are parameters randomly chosen for each node from a continuous uniform distribution (see Appendix H for details). This results in changes in rankings, and swaps in the order of ranks, reminiscent of real scenarios where teams and players rise and fall. The fact that we assign individual parameters to nodes allows us to mimic realistic scenarios where different teams change their ranks at different rates during a season. For instance, some teams can have more constant ranks while others can change more rapidly.

To assess the effect of different network structures, we vary parameters β and c from Eq. (7). We tabulate the results in Table I for varying values of β and fixed $c = 0.5$, and in Table V for varying values of c and fixed $\beta = 2.0$. We use 50% of the data for training and four time-steps for testing, detailed in Appendix F.

Overall, DSR has the largest number of top performances when considering all metrics (Tables I and V). Notably, DSR outperforms its offline variant OFFDSR, even though OFFDSR is given the entire history. This implies that using future interactions to retrodict out-of-sample interactions is less accurate than simply using past interactions. Recall also that DSR is more efficient algorithmically than OFFDSR. Overall, all algorithms perform better for higher values of β (i.e., lower noise).

The model with the second largest number of top performances is WHR, which does well particularly for σ_L, the metric that accounts for the likelihood of the outcomes. Notably, static SpringRank is significantly worse than the other models, illustrating that performance can be negatively affected by choosing a static model in dynamical settings. However, for higher noise levels such as $\beta = 0.1$, static SpringRank performs comparably well to the other models. This suggests that when there is less structure in the data, a static algorithm is enough: taking the chronological order of events into account does not improve performance.

As a sanity check of our permutation test for model selection between static and dynamic models, we also considered synthetic datasets generated with static ranks $s_i^t = s_i$. As expected, static SpringRank performs well in comparison to the dynamic algorithms as shown in Table VII.

Finally, we qualitatively investigate the inferred ranking in Fig. 2 for DSR, Elo, and W-L where the hierarchy as well as predictive performance is strong, as can be seen in Table I when $\beta = 2.0$. We notice how the time-scale of the evolution of the ranks is different in all cases, with W-L having frequent and sudden jumps while DSR and Elo are smoother with roughly equal performance. In all cases, though, we notice small jumps indicating changes in ranks that deviate from the smoothness in the ground truth. Nevertheless, performance is strong for DSR and Elo, who perform roughly equally well, as the behaviors of the individual trajectories resembles that of the ground truth well in both cases.

These synthetic tests suggest that dynamical algorithms capture relevant information when the data has a hierarchical structure and chronological ordering matters (i.e., low noise). In these cases, Dynamical SpringRank performs the best according to several metrics. For higher noise levels or static ranks, timestamp information is no longer relevant and static SpringRank performs well.

TABLE I. Results obtained from synthetic data with varying noise levels, represented by $\boldsymbol{\beta}$. Each value is the mean of 4 independent realizations of the noisy model. The green highlighted values are the top performances for the considered metric. Notably, some of the values in the same row appear identical but only a single value is highlighted. The reason for this is that the highlighted value is better by less than three decimal places. Table IV contains the standard error of the above values. σ_a and σ_L cannot be applied to the W-L model, so there are no values for the metrics.

β	Metric	Elo	OFFDSR	mwSR	DSR	SR	TS	W-L	WHR
0.1	Accuracy	0.545	0.544	0.533	**0.549**	0.540	0.548	0.511	0.549
	Agony	1.568	1.596	1.658	1.574	1.646	**1.551**	1.784	1.578
	σ_a	0.593	**0.594**	0.576	0.584	0.594	0.593	–	0.592
	σ_L	−1.426	−1.382	−1.389	**−1.378**	−1.382	−1.392	–	−1.389
0.5	Accuracy	0.700	0.700	0.698	0.700	0.652	**0.703**	0.620	0.701
	Agony	0.881	0.877	0.887	**0.877**	1.075	0.885	1.230	0.882
	σ_a	0.666	0.647	**0.708**	0.705	0.635	0.674	–	0.670
	σ_L	−1.344	−1.286	−1.165	−1.163	−1.263	−1.167	–	**−1.152**
1.0	Accuracy	0.810	**0.816**	0.810	0.810	0.713	0.808	0.721	0.811
	Agony	0.455	0.436	**0.429**	0.440	0.799	0.458	0.766	0.442
	σ_a	0.771	0.783	0.813	**0.813**	0.702	0.767	–	0.756
	σ_L	−1.143	−0.988	−0.848	−0.853	−1.149	−0.863	–	**−0.846**
1.5	Accuracy	**0.866**	0.862	0.863	0.864	0.752	0.865	0.772	0.863
	Agony	**0.260**	0.269	0.269	0.261	0.655	0.266	0.546	0.270
	σ_a	0.835	0.823	0.863	**0.866**	0.745	0.825	–	0.815
	σ_L	−0.883	−0.918	−0.671	−0.670	−1.128	−0.662	–	**−0.655**
2.0	Accuracy	0.898	0.898	0.898	**0.903**	0.772	0.900	0.803	0.900
	Agony	0.172	0.179	0.171	**0.163**	0.606	0.172	0.451	0.169
	σ_a	0.876	0.847	0.899	**0.901**	0.769	0.861	–	0.856
	σ_L	−0.673	−0.844	−0.492	−0.500	−1.088	−0.500	–	**−0.492**

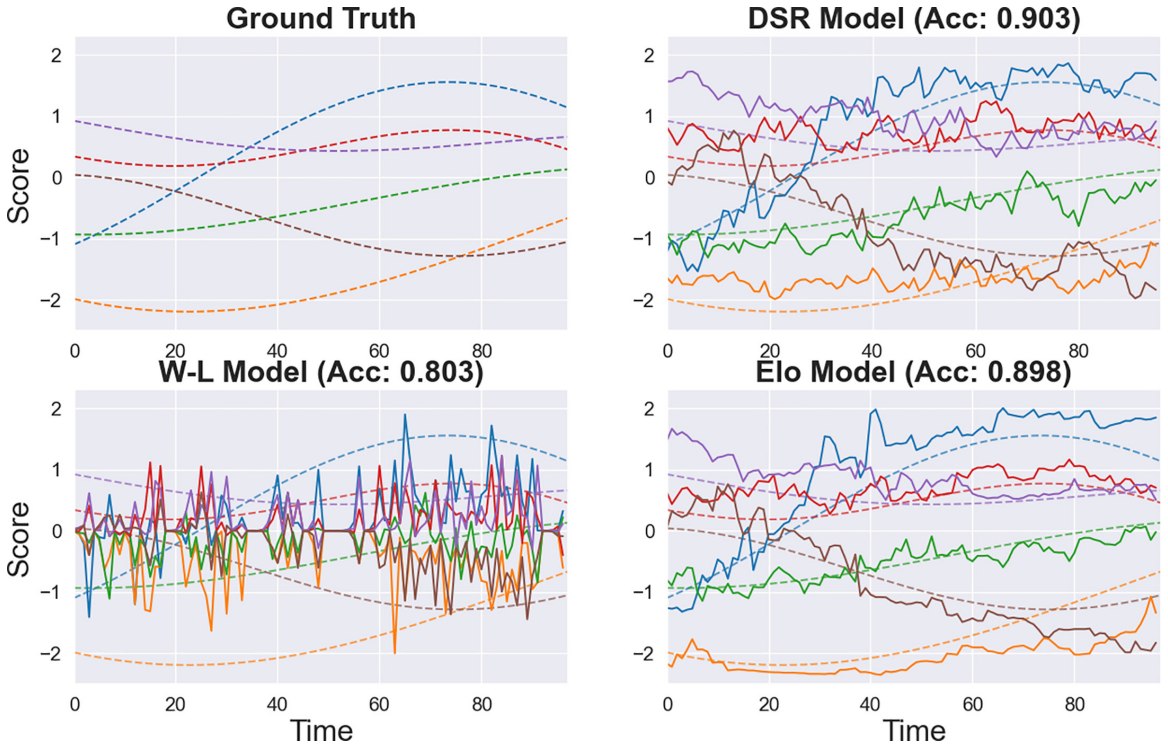

FIG. 2. Evolution of inferred ranks over time on synthetic data. We illustrate the inferred ranks of three models over time: DSR, W-L, and Elo. We also illustrate the ground truth of the synthetic ranks over time as a comparison (top left). The synthetic data is generated by setting $\beta = 2.0$ and $c = 0.5$. Dashed lines are ground truth ranks.

TABLE II. Descriptions of the real datasets.

Competition	Type	N_{teams}	N_{games}	T_{steps}
NBA	Basketball	30	9594	218
lichess.org	Chess	96	298	90
Serie A	Soccer	47	5679	397
English Premier League (EPL)	Soccer	39	3396	114

B. Performance on real data

We consider a variety of real datasets of timestamped interactions, as described in Table II. These datasets come from competitions in well-known sports such as soccer, basketball and chess. They are both relevant and relatable sources of information for our experiments.

In soccer, we consider the Italian Serie A and the English Premier League (EPL). The Serie A data is from the period 1993–2016 and contains the results of thousands of games between 47 teams. Similarly, the EPL contains results of thousands of games between 39 teams in the period 2006–2018. In contrast, the NBA dataset contains roughly three times the number of EPL matches from 2010–2018 between 30 teams. All three datasets can be found on *kaggle.com*. Finally, the chess dataset is obtained from matches on *lichess.org*. It contains 298 matches from 2014–2017. In all cases, A_{ij}^t is the number of times team i (or for chess, player i) beats j in a given time-step t. The definition of a time-step varies from sport to sport (see below).

As with synthetic data, we found that DSR outperforms the other algorithms in terms of the most top performances across our four different metrics (Table III). Elo and WHR are the next best performers: Elo does slightly better in a few cases on the accuracy or agony metric for NBA and chess, and as in the synthetic data WHR does well for the σ_L metric, the conditional log-likelihood of generating directed edges (outcomes) given their existence.

Perhaps surprisingly, static SpringRank performs well on both the Serie A and chess datasets, achieving the highest accuracy on Serie A. For Serie A, this could, in part, be explained by the fact that the dataset has a lower frequency of matches compared to the NBA. In a soccer competition, typically matches are played weekly, while in the NBA teams play more frequently, two or three times per week. It could be that a lower frequency implies fewer dependencies between time-steps, thus making a dynamical model that implies a dependence between time-steps less expressive. At the same time, the regulations behind the European soccer leagues and the NBA are quite different (with salary caps and college drafts aiming at levelling the teams' strength in the NBA). This could imply a more constant ranking in soccer than in the NBA, making a static model work well in practice. In fact, in the last 20 years in Serie A only four teams won the title and only six in the English Premier League, sometime with long winning streaks for an individual team. On the contrary, NBA championships are clearly more unpredictable, with ten different winners in last twenty years, with a maximum of two titles won consecutively by the same team.

For the chess dataset, each time-step represents a day of matches, but match days are not necessarily consecutive. For example, the first day of matches is 2014-03-04 and the second is 2015-11-15. Again, this poses the problem of large gaps in time which could lessen the connection between time-steps.

As such, in both the Serie A and chess datasets, it is understandable that the static version of SpringRank would perform fairly well as time-steps do not influence each other as much as in, for example, the NBA dataset. This is further supported by the closeness in results between the static version of SpringRank and the dynamic models on the soccer and chess datasets. In contrast, the gap of results from the NBA dataset between the aforementioned static and dynamic models is larger. We discuss the influence of time further in Sec. IV C. (The Serie A and chess datasets might also be

TABLE III. Results obtained from real data. The green highlighted values are the top performances for the considered metric. Notably, some of the values in the same row appear identical but only a single value is highlighted. The reason for this is that the highlighted value is better by less than three decimal places. Table IX contains the standard error of the above values. σ_a and σ_L cannot be applied to the W-L model hence there are no values for the metrics.

Dataset	Metric	Elo	OffDSR	mwSR	DSR	SR	TS	WHR	W-L
NBA	Accuracy	**0.650**	0.642	0.637	0.649	0.607	0.645	0.648	0.565
	Agony	**2.981**	3.050	3.084	2.987	3.568	3.006	2.997	4.071
	σ_a	0.579	0.562	0.639	**0.646**	0.596	0.584	0.580	–
	σ_L	−1.426	−1.330	−1.266	−1.256	−1.324	−1.280	**−1.255**	–
Chess	Accuracy	**0.677**	0.633	0.637	0.665	0.672	0.665	0.647	0.539
	Agony	8.404	11.641	9.242	8.470	8.074	7.693	8.179	**1.087**
	σ_a	0.615	0.580	0.626	**0.651**	0.581	0.628	0.626	–
	σ_L	−1.290	−1.341	−1.294	−1.333	−1.550	−1.255	**−1.206**	–
EPL	Accuracy	0.678	**0.681**	0.669	0.675	0.679	0.672	0.673	0.609
	Agony	3.239	4.144	4.141	**3.147**	3.401	3.797	3.825	5.438
	σ_a	0.595	0.530	0.669	**0.675**	0.662	0.601	0.598	–
	σ_L	−1.285	−1.357	−1.208	**−1.184**	−1.206	−1.211	−1.199	–
Serie A	Accuracy	0.655	0.652	0.630	0.653	**0.663**	0.655	0.653	0.564
	Agony	4.296	5.800	6.278	**4.041**	4.241	5.669	5.653	8.101
	σ_a	0.582	0.530	0.628	**0.652**	0.647	0.590	0.585	–
	σ_L	−1.363	−1.357	−1.287	−1.240	−1.269	−1.257	**−1.237**	–

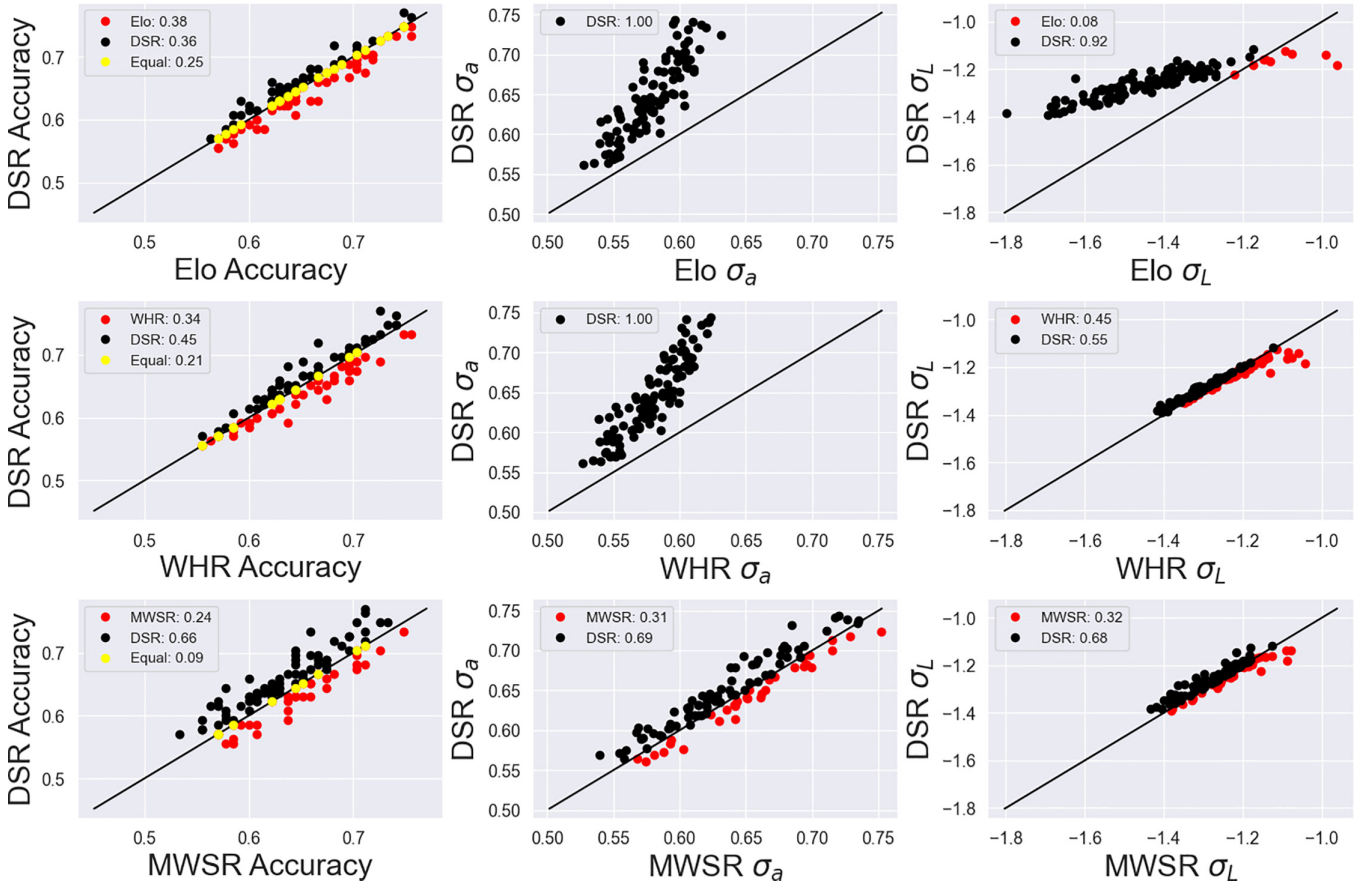

FIG. 3. Fold-by-fold evaluation on the NBA dataset. We compare the predictions of DSR to Elo, WHR and mwSR in relation to the performance metrics σ_a, σ_L and accuracy. The black points above the diagonal represent folds where DSR outperformed its competitors; yellow points indicate equal performance and red points represent DSR loses (where it was outperformed by competitors). Numbers inside the legend are the number of trials that an algorithm outperforms the other in percentage.

suitable for the model described above where time intervals between snapshots can vary; we leave this for future work.)

Overall, we observe a fairly broad distribution of values for the various metrics over the cross-validation trials, as there are matches that are more difficult to predict than others. Hence, we take a closer look by analyzing a fold-by-fold performance comparison, where we assess the number of test sets in which one algorithm outperforms the others. We find that DSR performs equal to or better than the other algorithms in most cases on the NBA dataset, and in all cases when compared to Elo and WHR in terms of σ_a (Fig. 3).

We observed qualitative differences of the inferred ranks in Fig. 4 similar to those observed in Fig. 2 for synthetic data. W-L infers ranks that change with a much higher frequency than the others. While smoother, the ranks inferred by TS show more frequent variations than DSR and Elo, which infer similarly behaving ranks.

C. Relevance of time

As a final consideration, we turn to a fundamental question: given a dataset of timestamped interactions, does their chronological order matter? If the answer is positive, then we should use a dynamical ranking algorithm to analyze the data. If not, then a simpler static algorithm should be enough.

One way to assess whether a given dataset is better modeled by a dynamical or static algorithm is by randomly permuting the order of interactions—but not their outcomes—and thus removing any relationship between ranks and time. If an algorithm performs significantly better on the original data than on the permuted data, then the order matters and a dynamical model is justified. To be more precise, applying random permutations to the data produces a distribution of any test statistic, including any measure of the performance of an algorithm that predicts which way a given interaction will go (e.g., which of two players will win a chess match, conditioned on the event that they play). If the performance on the original data is far out in the tail of this distribution, then we can reject the null hypothesis that the time-steps are simply independent draws from a static model.

We run this permutation test first on synthetic data, confirming as expected that the dynamical model performs significantly better on synthetic data generated with the time-varying model introduced in Sec. III B, provided that the hierarchy itself is sufficiently strong (Fig. 10). However, when the hierarchy is weak (i.e., β is small), the ranks have little relationship to the outcomes, and treating the ranks dynamically is no longer justified by the permutation tests (Fig. 10).

For NBA data, permutation tests show that chronological order matters, and that using a dynamical model significantly

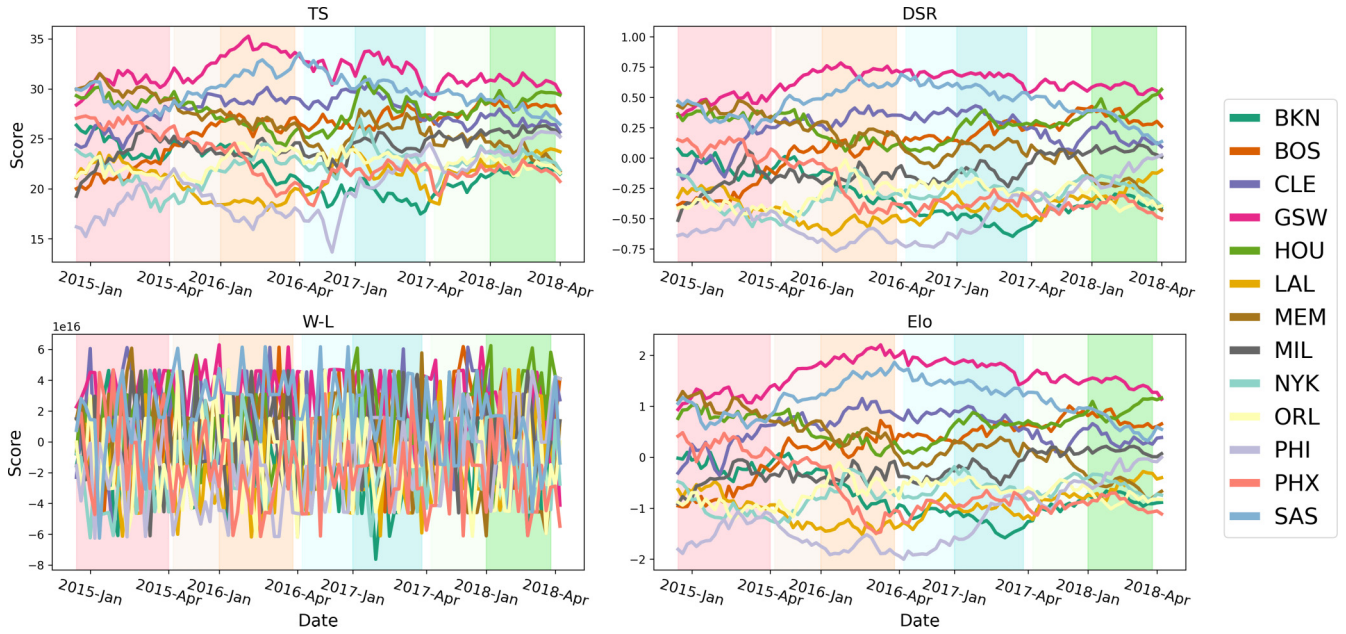

FIG. 4. Evolution over time of predicted ranks for the NBA dataset. We illustrate the predicted ranks of four models over time: TS, DSR, W-L and Elo. We select a subset of 13 teams (as indicated in the legend) to highlight the behaviors of both top and bottom scoring teams. Vertical colored bands break seasons into two periods.

improves prediction (Fig. 5). However, for the soccer and chess datasets, we find mixed results depending on the test statistic. For instance, the "agony" (a measure which penalizes the model for interactions $i \to j$ if $s_j - s_i$ is large) suggests that time-order is relevant, while the accuracy (the fraction of interactions whose direction is correctly predicted) is less sensitive to this information (Fig. 9, Table X). While the most straightforward explanation is that NBA rankings are more time-varying, while soccer and chess are less so, we also note that there are many more games in a NBA season than in a soccer season, since there are more teams and more frequent games in the NBA, therefore allowing our simple permutation test to reject the null hypothesis more easily with more available data to differentiate time-varying versus static ranks (Table II).

V. CONCLUSION

Dynamical SpringRank is a principled extension of the physics-inspired SpringRank model for dynamic hierarchal structures, which lets us infer time-varying ranks from timestamped interactions. By coupling individuals' previous and current ranks, it exploits the chronological ordering of the data to better predict the outcomes of future interactions. It contains a parameter k that can be tuned or learned to control the smoothness of the change in ranks, or equivalently the weight given to past ranks.

We constructed two different formulations of Dynamic SpringRank: an online and an offline one, which are given just past ranks and the entire history, respectively. The online version performed better and is less computationally expensive. However, both models, similar to the static version, are

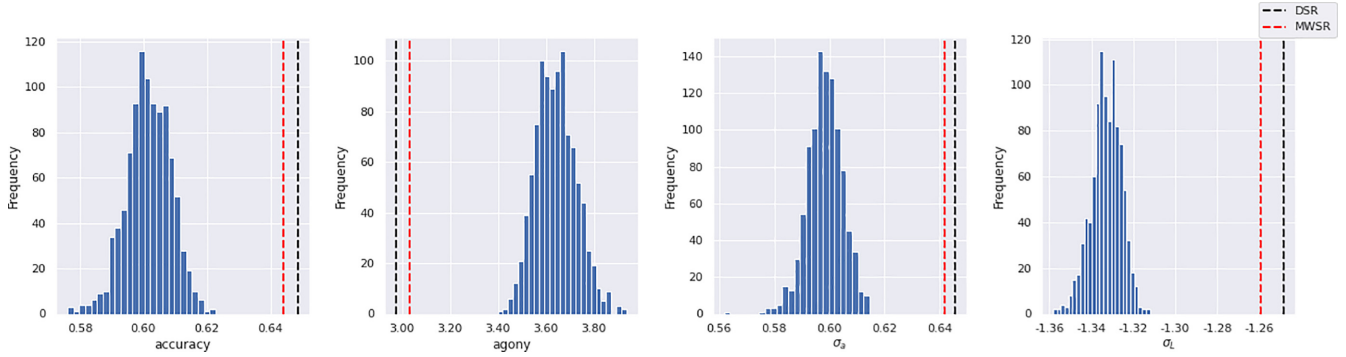

FIG. 5. Permutation test results on the NBA dataset: chronology matters. The histogram is generated by 1000 random permutations to the NBA dataset, and measuring the performance of Dynamical SpringRank on these permuted datasets. The black and red dotted lines represent the results of DSR and mwSR, respectively, on the original, chronologically ordered NBA dataset—the accuracy is much higher, and the agony much lower, than the vast majority of permuted datasets. This convincingly rejects the null hypothesis that chronological order does not matter, and justifies the use of a dynamical model. In each case the p-value is less than 0.001.

scalable algorithms that require sparse linear algebra and provide a probabilistic generative model for creating dynamically directed networks with tunable levels of hierarchy and sparsity.

We also illustrated that in dynamic settings where time information is important, Dynamical SpringRank is better than its static counterpart. Its ability to predict future outcomes in dynamical settings proved to be similar or better than other state-of-the-art dynamical ranking algorithms for a variety of metrics and datasets, both synthetic and real. An open-source implementation of both *offline* and *online* versions of Dynamical SpringRank is available at Ref. [30].

For future work, we defined more elaborate models where the time intervals between interactions can vary, or where a momentum term induces smoothness in the rate at which ranks change over time. Another (perhaps challenging) direction is to couple the rank dynamics with the entities' choices to interact with each other. For instance, one can imagine a model in which animals tend to challenge those immediately above them in the dominance hierarchy, or where new arrivals to a community test themselves against current members to find their place, or even three-way interactions where an animal who attacks another is punished by a third [31]. Testing these models would require rich data from biological and social systems.

ACKNOWLEDGMENT

We thank Jean-Gabriel Young for his feedback on the boundary conditions of the Offline Dynamical SpringRank model.

APPENDIX A: FULL DERIVATION SELF-SPRING INTERACTION

Calculate the ith component of the gradient:

$$\begin{aligned}\frac{\partial H_{\text{total}}}{\partial s_i^t} &= \sum_j \left[A_{ij}^t(s_i - s_j - \ell_0) - A_{ji}^t(s_j - s_i - \ell_0)\right] + k\left(s_i^t - s_i^{t-1}\right)\\ &= \sum_j \left(A_{ij}^t + A_{ji}^t\right) s_i^t - \sum_j \left(A_{ij}^t + A_{ji}^t\right) s_j^t - \sum_j \left(A_{ij}^t - A_{ji}^t\right)\ell_0 + k\left(s_i^t - s_i^{t-1}\right)\\ &= \left(d_i^{\text{out},t} + d_i^{\text{in},t} + k\right) s_i^t - \sum_j \left(A_{ij}^t + A_{ji}^t\right) s_j^t - \left(d_i^{\text{out},t} - d_i^{\text{in},t}\right)\ell_0 - k\, s_i^{t-1}.\end{aligned}$$

Imposing $\nabla H = 0$ we obtain

$$\left(d_i^{\text{out},t} + d_i^{\text{in},t} + k\right) s_i^t - \sum_j \left(A_{ij}^t + A_{ji}^t\right) s_j^t = \left(d_i^{\text{out},t} - d_i^{\text{in},t}\right)\ell_0 + k\, s_i^{t-1},$$

which yields

$$[D^{\text{out},t} + D^{\text{in},t} - (A^t + (A^t)^\dagger) + k\mathbb{I}]\,\mathbf{s}^{t,*} = [D^{\text{out},t} - D^{\text{in},t}]\ell_0 + k_0\,\mathbf{s}^{t-1},$$

as reported in Eq. (5) for $\ell_0 = 1$.

APPENDIX B: FULL DERIVATION OF SELF-SPRING INTERACTION OVER ALL TIME

Calculate the ith component of the gradient:

$$\begin{aligned}\frac{\partial H_{\text{total}}}{\partial s_i^t} &= \frac{\partial}{\partial s_i^t}\left[\sum_t^T H^t(\boldsymbol{s}^t, \boldsymbol{s}^{t-1})\right]\\ &= \frac{\partial}{\partial s_i^t}\left[\sum_t^T \sum A_{ij}^t H_{ij}\left(s_i^t, s_j^t\right)\right] + \frac{\partial}{\partial s_i^t}\left[\sum_t^T \sum k H_{\text{self}}\left(s_i^t, s_i^{t-1}\right)\right]\\ &= \sum_j^N \left[A_{ij}^t\left(s_i^t - s_j^t - \ell_0\right) - A_{ji}^t\left(s_j^t - s_i^t - \ell_0\right)\right] - k s_i^{t-1} + 2k s_i^t - k s_i^{t+1}\\ &= \sum_j \left(A_{ij}^t + A_{ji}^t\right) s_i^t - \sum_j \left(A_{ij}^t + A_{ji}^t\right) s_j^t - \sum_j \left(A_{ij}^t - A_{ji}^t\right)\ell_0 - k s_i^{t-1} + 2k s_i^t - k s_i^{t+1}\\ &= \left(d_i^{\text{out},t} + d_i^{\text{in},t} + k\right) s_i^t - \sum_j \left(A_{ij}^t + A_{ji}^t\right) s_j^t - \left(d_i^{\text{out},t} - d_i^{\text{in},t}\right)\ell_0 - k\, s_i^{t-1} + 2k s_i^t - k s_i^{t+1}.\end{aligned}$$

Imposing $\nabla H = 0$ we obtain

$$\left(d_i^{\text{out},t} + d_i^{\text{in},t} + 2k\right) s_i^t - \sum_j \left(A_{ij}^t + A_{ji}^t\right) s_j^t = \left(d_i^{\text{out},t} - d_i^{\text{in},t}\right)\ell_0 + k\left(s_i^{t-1} + s_i^{t+1}\right),$$

which yields

$$[D^{\text{out},t} + D^{\text{in},t} - (A^t + (A^t)^\dagger) + 2k\mathbb{I}]s^{t,*} = [D^{\text{out},t} - D^{\text{in},t}]\ell_0 + k_0(s^{t-1} + s^{t+1}),$$

as reported in Eq. (6) for $\ell_0 = 1$.

APPENDIX C: DYNAMIC SPRING REST LENGTH

As an alternative to the time-dependency presented in the main text (i.e., through self-springs), we also investigated the introduction of a time-dependent rest length. In this case we assume a dynamic rest length ℓ_{ij}^t for the interaction at time t between i and j. To enforce a relationship between current and past ranks, we assume ℓ_{ij}^t to be a function of the rank difference $s_i^{t-1} - s_j^{t-1}$ between i and j at time $t-1$:

$$H_{ij}^t(s_i^t, s_j^t) = \tfrac{1}{2}\left(s_i^t - s_j^t - l_{ij}^t\right)^2,$$

where

$$l_{ij}^t = s_i^{t-1} - s_j^{t-1} + \ell_0. \tag{C1}$$

The resultant Hamiltonian for the whole system is

$$H^t(\mathbf{s}^t, \mathbf{s}^{t-1}) = \sum_{i,j} A_{ij}^t H_{ij}^t(s_i^t, s_j^t).$$

As opposed to Eq. (4), here we do not have self-interactions. Instead, past ranks appear directly inside the rest lengths. If we define a new variable $z_i^t = s_i^t - s_i^{t-1}$, then we obtain the Hamiltonian

$$H^t(\mathbf{z}^t) = \sum_{i,j} A_{ij}^t H_{ij}^t(z_i^t, z_j^t) = \sum_{i,j} \frac{A_{ij}^t}{2}\left(z_i^t - z_j^t - \ell_0\right)^2,$$

which is the same Hamiltonian used in static SpringRank [1] but as a function of the auxiliary variable $\mathbf{z}^t$. Thus, we know that the ground state $\mathbf{z}^{t,*}$ will be the solution of the linear system:

$$[D^{\text{out}} + D^{\text{in}} - (A + A^\dagger)]\mathbf{z}_t^* = [D^{\text{out}} - D^{\text{in}}]\ell_0\mathbf{1}.$$

The idea is that once $\mathbf{z}^{t,*}$ is obtained by solving this linear system, one can extract the ranks as $s_i^t = z_i^t + s_i^{t-1}$, where s_i^{t-1} is known from the inference of the previous step. Notice that in the extreme case of having only two individuals i, j, initializing $s_i^0 = s_j^0 = 0$ and i as the constant winner ($A_{ij}^t \geqslant 0$ and $A_{ji}^t = 0\,\forall t$), we would infer $s_i^1 - s_j^1 = \ell_0$ at the first time-step. Then iterating in time yields $\ell_{ij}^t = t\ell_0$. In words, for situations where the hierarchy is strong and time is constant (i.e., a stronger individual always defeats a weaker one at any time-step), the rest length would grow linearly in time. As a consequence, the distance between ranks grows further and further, driving them apart. This is the case in sports, for instance, where teams earn points for each win, distancing them more and more from the losing teams. In other situations, we might want instead a scenario where the difference between ranks becomes a constant value ℓ_0 the more we collect consistent observations in time, i.e., $\forall t$, $s_i^t - s_j^t = \ell_0$. This can be easily obtain by changing the model's details, like setting a different initial rest length and update in Eq. (C1).

APPENDIX D: PERFORMANCE EVALUATION

In this section, we discuss the various metrics used in more detail. Accuracy is a coarse-grained measure to evaluate the quality of predictions. It is the fraction of times an observed directed edge points from the higher towards the lower ranked node, i.e., the number of times that a *stronger* (according to our ranking) individual *beats* a weaker one,

$$\text{accuracy} = \frac{1}{M}\sum_{i,j} A_{ij}\,\Theta(s_i - s_j),$$

where $\Theta(x) = 1$ if $x > 0$, $\Theta(x) = 0.5$ if $x = 0$ and $\Theta(x) = 0$ if $x < 0$; $M = \sum_{i,j} A_{ij}$.

If we call an *upset* an interaction where a lower ranked individual beats someone stronger, then the accuracy is just 1 minus the fraction of upsets. Accuracy does not weigh upsets differently. However, in certain situations making an erroneous prediction involving individuals nearby in rank might be less important than an error involving individuals far in rank. In this case, it is useful to consider the *agony* function [7]. It considers the difference in ordinal ranks as penalties.[1] Subsequently, an upset between two nodes close in rank counts much less than an upset between two nodes far rank, based on a parameter d:

$$\text{agony} = \frac{1}{M}\sum_{i,j} A_{ij}\,\max(0, r_i - r_j)^d,$$

where $r_i \in [0, .., n-1]$ is the *ordinal* rank of node i (which can naturally be extracted from the real-valued ranks s_i). When $d = 0$ we recover the standard number of unweighted upsets. We use $d = 1$ in our evaluation of models. The more the rank is informative towards the predicted outcomes, the lower the value of the agony and the less the hierarchy is violated.

Accuracy and agony are metrics for ordinal rankings. For real-valued models such as SpringRank, it is worth considering fine-grained metrics as well. We thus consider in our experiments two other metrics that take into account an estimate of P_{ij}—the probability that i beats j.

First, σ_a is the average probability assigned to the correct direction of an edge:

$$\sigma_a = 1 - \frac{1}{2M}\sum_{ij} |A_{ij} - \overline{A_{ij}}P_{ij}|,$$

where $\overline{A_{ij}} = A_{ij} + A_{ji}$ is the number of interactions between i and j.

[1] We use *positional* ranks instead of the real-valued ranks to avoid scale problems comparing different algorithms.

Second, σ_L is the conditional log-likelihood of generating the directed edges *given* their existence:

$$\sigma_L = \log P(A|\bar{A}) = \sum_{ij} \binom{A_{ij} + A_{ji}}{A_{ij}} + \log[P_{ij}(\beta)^{A_{ij}}(1 - P_{ij}(\beta))^{A_{ji}}].$$

Notice that we explicitly highlight the dependence of P_{ij} on the (inverse) *temperature* parameter β which control the level of hierarchy in the predictions. For $\beta \to \infty$ the network is fully hierarchical which means that an edge between i and j, with $s_i > s_j$, points from $i \to j$ with $P_{ij} = 1$. In contrast, when $\beta = 0$, the predicted outcomes are completely random with $P_{ij} = P_{ji} = 0.5$.

In general, maximizing σ_a and σ_L requires two distinct values for β that we will denote as $\hat{\beta}_a$ and $\hat{\beta}_L$. Intuitively, the reason is that a single severe mistake where $A_{ij} = 1$ but $P_{ij} \approx 0$ reduces the likelihood by a large amount, while only reducing the accuracy by one edge. As a result, predictions using $\hat{\beta}_a$ produce fewer incorrectly oriented edges and achieve a higher σ_a on the test set. However, predictions using $\hat{\beta}_L$ will produce fewer dramatically incorrect predictions where P_{ij} is very low, and thus achieve higher σ_L on the test set [1]. In other words, a prediction model that maximizes σ_L tends to be more cautious in assigning high probabilities of *success*, even in very unbalanced matches, to avoid potential impactful mistakes. In contrast, a model optimizing σ_a can be less conservative, ignoring isolated (even dramatic) mistakes and favoring a good frequency of predictions as close as possible to the real probability.

APPENDIX E: DESCRIPTION OF ALGORITHMS USED FOR COMPARISON

1. Elo rating system [21]

This method assumes an hidden score R_i for each node i. The expected score S_{ij} of a game between players i and j is a function of the score difference $R_i - R_j$ as

$$S_{ij}(R_i, R_j) = \frac{1}{1 + 10^{-(R_i - R_j)/400}}. \tag{E1}$$

The actual score A_{ij} of the game is 1 if player i wins, 1/2 if the game is a draw, and 0 if player i loses. After observing the outcome of the match, the score of i is updated according to the following rule:

$$R_i^{\text{new}} = R_i + K(A_{ij} - S_{ij}), \tag{E2}$$

where K is an attenuation factor that determines the weight that should be given to a player's performance relative to their previous rating. We used grid-search to determine K. The above formula has a natural interpretation. The term $A_{ij} - S_{ij}$ represents a discrepancy between what was expected and what was observed. If this term is positive, then the player achieved a result better than what predicted by the rating at the previous time step. Hence, the player's rating is increased to reflect the possible improvement in strength. Similarly, if the term $A_{ij} - S_{ij}$ is negative, then the player performed worse than expected. Hence, this player's rating decreases by the discrepancy magnified by the value K.

2. WHR system [28]

This algorithm is based on the dynamic Bradley-Terry model [32]. The Bradley-Terry model for paired comparisons

TABLE IV. Standard error of results from synthetic data with varying noise levels, represent by β.

β	Metric	Elo	OFFDSR	mwSR	DSR	SR	TS	W-L	WHR
0.1	Accuracy	0.0073	0.0065	0.0067	0.0070	0.0064	0.0068	0.0068	0.0077
	Agony	0.0286	0.0269	0.0278	0.0267	0.0331	0.0265	0.0325	0.0284
	σ_a	0.0028	0.0031	0.0044	0.0039	0.0032	0.0029	–	0.0028
	σ_L	0.0107	0.0008	0.0055	0.0044	0.0019	0.0077	–	0.0066
0.5	Accuracy	0.0056	0.0054	0.0057	0.0056	0.0042	0.0051	0.0074	0.0054
	Agony	0.0197	0.0196	0.0202	0.0203	0.0168	0.0190	0.0314	0.0185
	σ_a	0.0027	0.0030	0.0051	0.0048	0.0031	0.0029	–	0.0029
	σ_L	0.0215	0.0035	0.0127	0.0114	0.0061	0.0121	–	0.0110
0.1	Accuracy	0.0056	0.0053	0.0048	0.0050	0.0073	0.0047	0.0060	0.0044
	Agony	0.0164	0.0156	0.0152	0.0161	0.0297	0.0141	0.0218	0.0142
	σ_a	0.0035	0.0037	0.0044	0.0047	0.0059	0.0030	–	0.0029
	σ_L	0.0318	0.0079	0.0153	0.0159	0.0254	0.0151	–	0.0135
0.5	Accuracy	0.0049	0.0050	0.0050	0.0048	0.0052	0.0049	0.0062	0.0046
	Agony	0.0126	0.0132	0.0124	0.0131	0.0210	0.0129	0.0243	0.0118
	σ_a	0.0035	0.0040	0.0046	0.0043	0.0047	0.0037	–	0.0034
	σ_L	0.0346	0.0080	0.0203	0.0232	0.0263	0.0187	–	0.0148
2.0	Accuracy	0.0038	0.0042	0.0036	0.0040	0.0053	0.0047	0.0056	0.0046
	Agony	0.0092	0.0095	0.0080	0.0082	0.0206	0.0094	0.0183	0.0089
	σ_a	0.0032	0.0034	0.0035	0.0038	0.0051	0.0034	–	0.0031
	σ_L	0.0301	0.0073	0.0152	0.0144	0.0302	0.0163	–	0.0147

TABLE V. Results obtained from synthetic data with varying density levels, represented by $\boldsymbol{c}$. Each value is the mean of 4 independent realizations of the model. The green highlighted values are the top performances for the considered metric. Notably, some of the values in the same row appear identical but only a single value is highlighted. This is because the highlighted value is better by less than three decimal places. Table VI contains the standard error for the above values. σ_a and σ_L cannot be applied to the W-L model hence there are no values for the metrics.

c	Metric	Elo	OFFDSR	mwSR	DSR	SR	TS	W-L	WHR
1.0	Accuracy	0.905	0.901	0.903	0.904	0.774	**0.905**	0.845	0.903
	Agony	0.161	0.167	0.163	0.165	0.562	**0.159**	0.299	0.162
	σ_a	0.895	0.892	0.909	**0.910**	0.778	0.889	–	0.884
	σ_L	−0.574	−0.705	−0.459	−0.459	−1.054	−0.451	–	**−0.450**
1.5	Accuracy	0.900	0.897	0.902	**0.903**	0.770	0.902	0.852	0.903
	Agony	0.161	0.171	0.159	**0.157**	0.608	0.162	0.280	0.157
	σ_a	0.899	0.902	0.908	**0.909**	0.771	0.894	–	0.895
	σ_L	−0.581	−0.617	−0.462	−0.459	−1.155	−0.460	–	**−0.452**
2.0	Accuracy	**0.909**	0.905	0.904	0.907	0.768	0.904	0.874	0.904
	Agony	**0.147**	0.156	0.154	0.148	0.601	0.154	0.217	0.153
	σ_a	0.911	0.910	0.915	**0.916**	0.772	0.907	–	0.902
	σ_L	−0.579	−0.575	−0.464	−0.453	−1.152	**−0.452**	–	−0.453
2.5	Accuracy	0.904	0.904	0.905	**0.906**	0.763	0.905	0.877	0.905
	Agony	0.159	0.160	0.159	**0.155**	0.601	0.160	0.216	0.160
	σ_a	0.912	0.913	0.916	**0.918**	0.773	0.909	–	0.907
	σ_L	−0.601	−0.650	−0.470	−0.463	−1.172	−0.459	–	**−0.458**
3.0	Accuracy	**0.910**	0.908	0.908	0.909	0.768	0.909	0.886	0.909
	Agony	**0.147**	0.150	0.152	0.149	0.605	0.152	0.198	0.151
	σ_a	0.921	0.921	0.920	**0.921**	0.776	0.917	–	0.915
	σ_L	−0.549	−0.559	−0.455	−0.447	−1.158	−0.438	–	**−0.438**

TABLE VI. Standard error of results from synthetic data with varying density, represented by c.

c	Metric	Elo	OFFDSR	mwSR	DSR	SR	TS	W-L	WHR
1.0	Accuracy	0.0021	0.0025	0.0023	0.0024	0.0040	0.0024	0.0040	0.0022
	Agony	0.0044	0.0055	0.0048	0.0050	0.0138	0.0052	0.0093	0.0051
	σ_a	0.0017	0.0023	0.0020	0.0020	0.0031	0.0021	–	0.0021
	σ_L	0.0171	0.0087	0.0099	0.0099	0.0200	0.0106	–	0.0095
1.5	Accuracy	0.0026	0.0027	0.0025	0.0023	0.0039	0.0026	0.0034	0.0026
	Agony	0.0051	0.0056	0.0053	0.0050	0.0180	0.0057	0.0086	0.0056
	σ_a	0.0016	0.0019	0.0021	0.0020	0.0034	0.0016	–	0.0017
	σ_L	0.0151	0.0053	0.0084	0.0085	0.0296	0.0086	–	0.0085
2.0	Accuracy	0.0019	0.0021	0.0020	0.0019	0.0032	0.0022	0.0025	0.0023
	Agony	0.0047	0.0044	0.0049	0.0046	0.0130	0.0050	0.0052	0.0052
	σ_a	0.0015	0.0018	0.0019	0.0018	0.0026	0.0016	–	0.0016
	σ_L	0.0176	0.0079	0.0095	0.0087	0.0273	0.0095	–	0.0089
2.5	Accuracy	0.0017	0.0016	0.0016	0.0017	0.0036	0.0017	0.0021	0.0018
	Agony	0.0042	0.0040	0.0040	0.0039	0.0140	0.0045	0.0044	0.0046
	σ_a	0.0014	0.0013	0.0015	0.0014	0.0030	0.0014	–	0.0014
	σ_L	0.0143	0.0065	0.0076	0.0073	0.0285	0.0076	–	0.0072
3.0	Accuracy	0.0015	0.0018	0.0016	0.0017	0.0027	0.0017	0.0022	0.0016
	Agony	0.0031	0.0037	0.0036	0.0035	0.0146	0.0035	0.0051	0.0037
	σ_a	0.0010	0.0013	0.0014	0.0013	0.0025	0.0010	–	0.0010
	σ_L	0.0102	0.0046	0.0061	0.0059	0.0278	0.0055	–	0.0054

TABLE VII. Results obtained from synthetic data in a static framework. Performance comparison of the various models on a synthetic dataset where the ranks are fixed along time (static framework). The green highlighted values are the top performances for the considered metric. Table VIII contains the standard error of the above values. σ_a and σ_L cannot be applied to the W-L model, so there are no values for the metrics.

Metric	Elo	OFFDSR	mwSR	DSR	SR	TS	W-L	WHR
Accuracy	0.715	0.716	0.696	0.720	0.722	**0.724**	0.552	0.723
Agony	0.528	0.518	0.565	0.523	**0.498**	0.517	1.099	0.515
σ_a	0.666	0.618	0.704	**0.733**	0.669	0.687	–	0.679
σ_L	−1.211	−1.324	−1.231	−1.172	−1.148	**−1.109**	–	−1.111

assumes that each node i has a rating $\gamma_i(t) = 10^{R_i(t)/400}$, where $R_i(t)$ is the Elo rating of player i at time t. Based on this, the probability of i winning a game against j at time t is

$$P\left(A_{ij}^t > 0 | \gamma_i, \gamma_j\right) = \frac{\gamma_i(t)}{\gamma_i(t) + \gamma_j(t)}. \tag{E3}$$

The WHR algorithm consists in estimating the values of $\gamma(t)$ using posterior inference of $p(\gamma|A)$ via Bayes' rule using the following expression:

$$p(\gamma|A) = \frac{P(A|\gamma)\, p(\gamma)}{P(A)}, \tag{E4}$$

where $p(\gamma)$ is a prior distribution on γ, $P(A)$ is a normalizing constant and $P(A|\gamma)$ is the Bradley-Terry model described in Eq. (E3).

3. Dynamic win-lose score [25]

This method assumes two scores for each node i at any given time step t_n, a win score $w_{t_n,i}$ and a loss score $\ell_{t_n,i}$. Let A_t be the win-loss matrix for the game that occurs at time t_n $(1 \leqslant n \leqslant n_{\max})$. If player j wins against player i at time t_n, then the (i, j) element of the matrix A_{t_n} is set to 1. All the other elements of A_{t_n} are set to 0. The method accounts for the effect of wins or losses by using a discounted past history and indirect results, i.e., results involving players that compete against a common opponent. Formally, it defines a "win" matrix W_{t_n} as follows:

$$\begin{aligned} W_{t_n} = A_{t_n} &+ e^{-\beta(t_n - t_{n-1})} \sum_{m_n \in \{0,1\}} \alpha^{m_n} A_{t_{n-1}} A_{t_n}^{m_n} \\ &+ e^{-\beta(t_n - t_{n-2})} \sum_{m_{n-1}, m_n \in \{0,1\}} \alpha^{m_{n-1}+m_n} A_{t_{n-2}} A_{t_{n-1}}^{m_{n-1}} A_{t_n}^{m_n} \\ &+ \cdots + e^{-\beta(t_n - t_1)} \sum_{m_2, \ldots, m_n \in \{0,1\}} \alpha^{\sum_{i=2}^{n} m_i} A_{t_1} A_{t_2}^{m_2} \cdots A_{t_n}^{m_n}, \end{aligned}$$

where α is the weight of an indirect win and $\beta \geqslant 0$ represents the decay rate of the score in time. These are the two main hyperparameters of this model, we fix them using cross-validation. The first term A_{t_n} on the right-hand side represents the effect of the direct win at time t_n. The second term consists of two contributions. For $m_n = 0$, the quantity inside the sum represents the direct win at time t_{n-1}, which results in weight $e^{-\beta(t_n - t_{n-1})}$, discounted depending on β and the time passed between two time steps. For $m_n = 1$, the quantity represents the indirect win. The (i, j) element of $A_{t_{n-1}} A_{t_n}$ is positive if and only if player j wins against a player k at time t_n and k wins against i at time t_{n-1}. Player i gains score $e^{-\beta(t_n - t_{n-1})}\, \alpha$ out of this situation. For both cases $m_n = 0$ and $m_n = 1$,the jth column of the second term accounts for the effect of the j's win at time t_{n-1}. The other terms behave analogously considering also third order indirect interactions and so on. A similar matrix is defined to account for losses. Then, the win score $w_{t_n,i}$ of a player i is computed as the ith entry of the vector $w_{t_n} = W_{t_n}^T \mathbf{1}$, where $\mathbf{1}$ is the all-one vector. Similarly, one can get the loss score $\ell_{t_n,i}$ by considering the loss matrix.

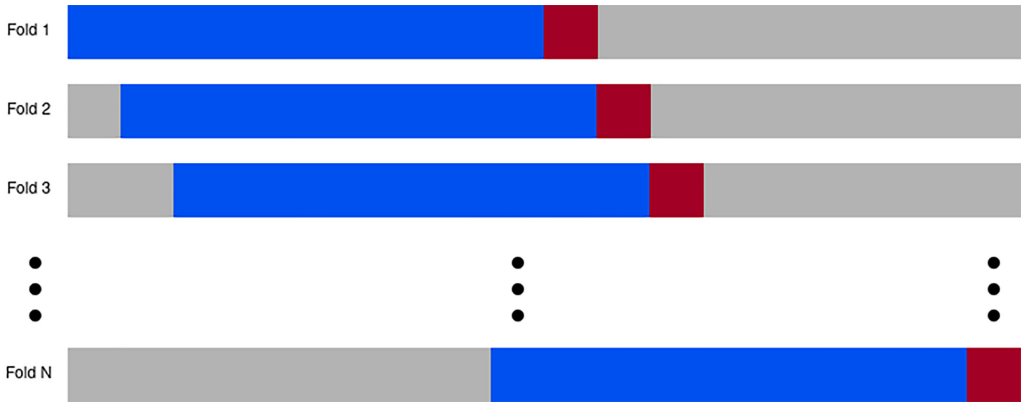

FIG. 6. Illustration of the cross-validation used in experiments. The blue bar represents the training set, the red bar is the test and the gray is the total dataset. N is the total number of folds.

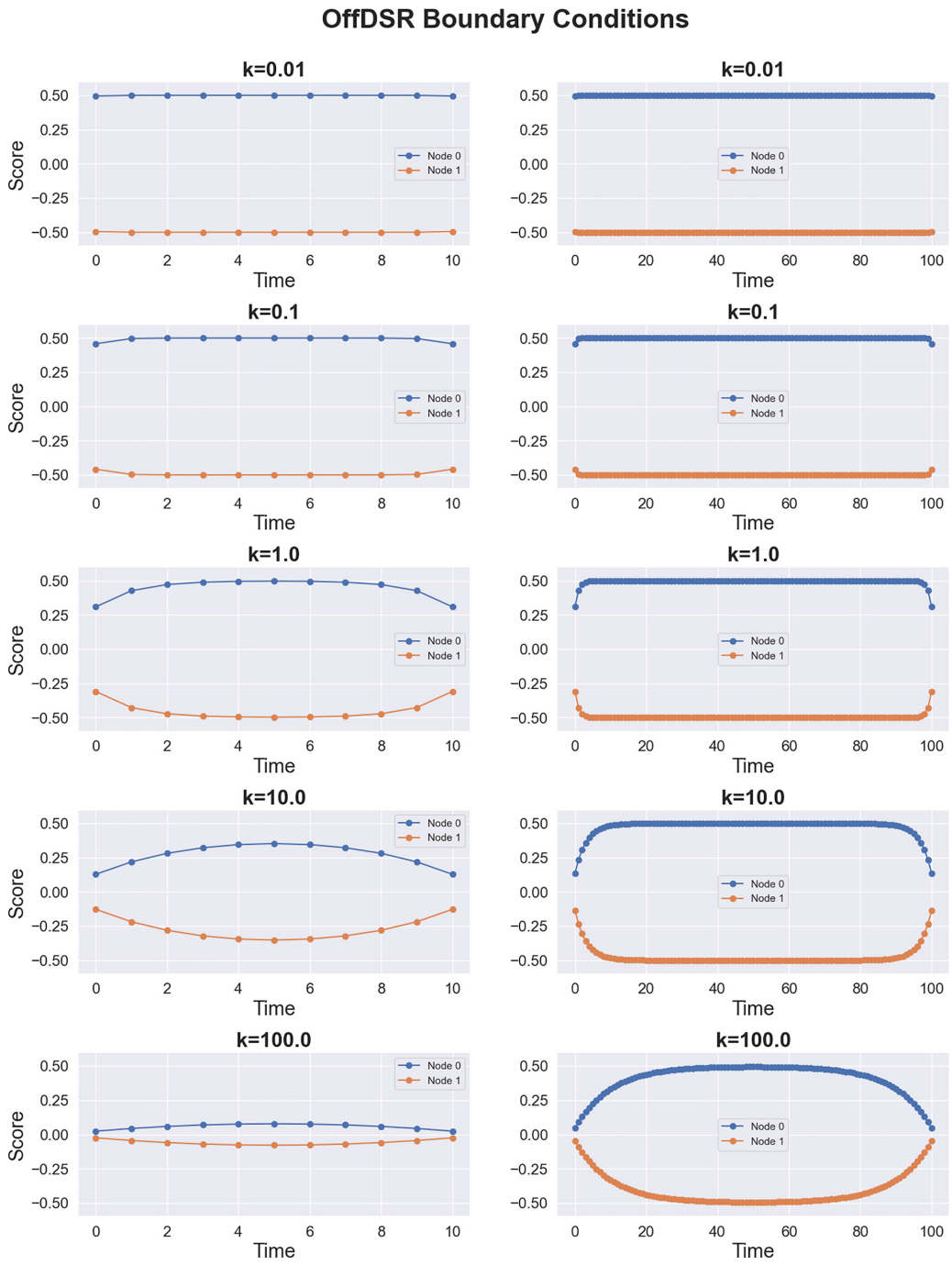

FIG. 7. OFFDSR with boundary conditions set to zero. A toy example was used to generate the plot where two nodes interact with a single directed edge pointing in the same direction for every timestep. The parameter k of OFFDSR is varied as well as the number of timesteps to further illustrate the effect of the boundary conditions on the ranks.

The final score for a player i at time t_n is the difference $s_{t_n,i} = w_{t_n,i} - \ell_{t_n,i}$.

4. TrueSkill rating system [27]

TrueSkill's current belief about a player's skill $s_{i,t}$ at time t is represented by a Gaussian distribution with mean μ_i and variance σ_i^2. This is inferred using Bayesian inference, where the goal is to estimate the posterior distribution $P(s_{i,t}|r)$, where r is a vector containing the rank of the nodes as determined by the outcomes, i.e., is a quantity determined by input data. The influence of the skill at the previous time step enters as a Gaussian prior centered at the value of the skill at the previous time step, i.e., $P(s_{i,t}|s_{i,t-1}, \gamma^2) = \mathcal{N}(s_{i,t}; s_{i,t-1}, \gamma^2)$, where γ is an hyperparameter. This method estimates the posterior distributions $P(s_{i,t}|r)$ with a Bayesian inference procedure that performs a Gaussian filtering that repeatedly smoothes the scores forward and backward in time, using approximate message-passing.

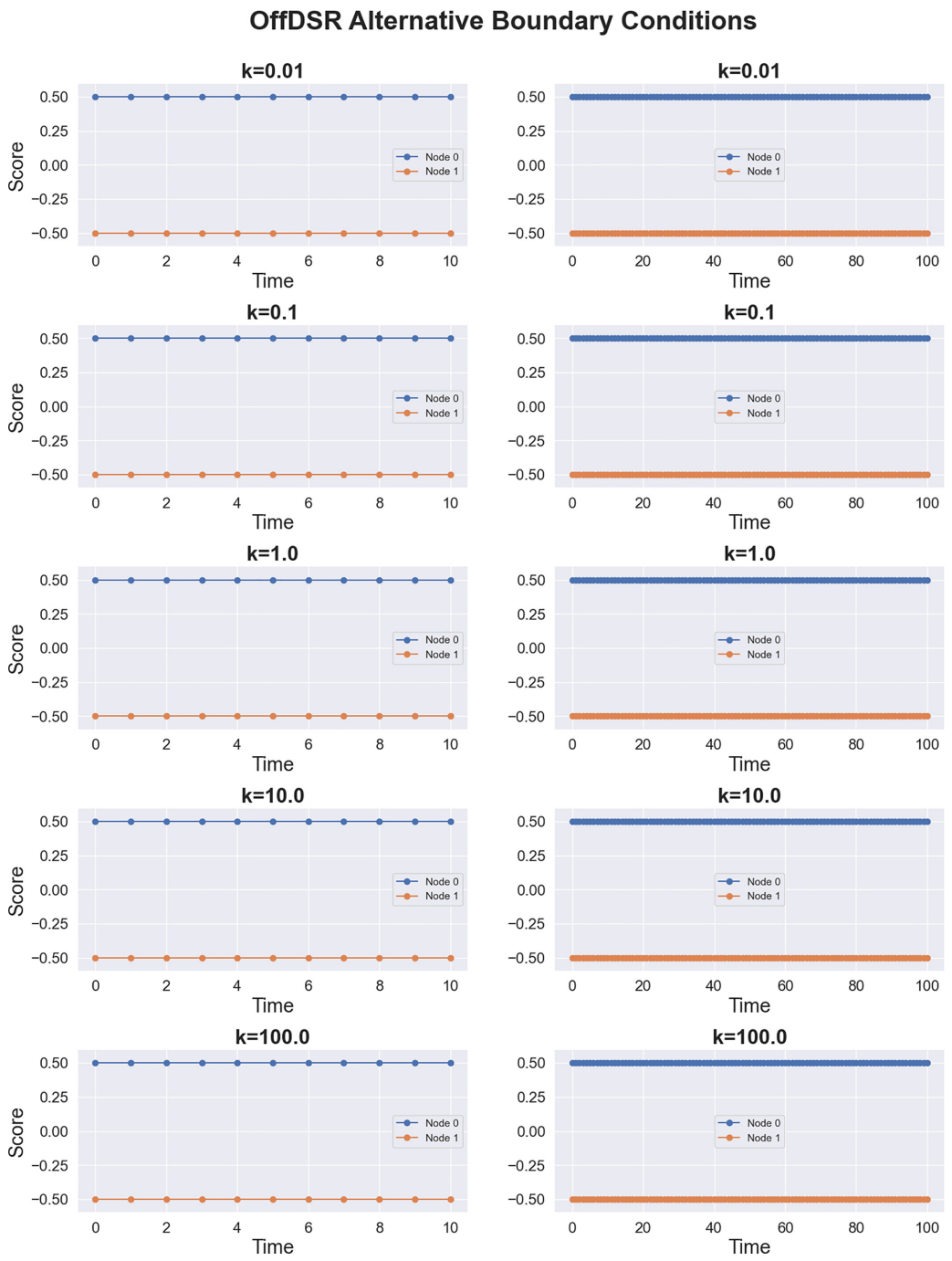

FIG. 8. OFFDSR with alternative boundary conditions. The first boundary condition is $\mathbf{s}^{t-1} = \mathbf{s}^t$ where $t = 0$ and the second boundary condition is $\mathbf{s}^t = \mathbf{s}^{t+1}$where $t = T$. A toy example was used to generate the plot where two nodes interact with a single directed edge pointing in the same direction for every timestep. This is the same as in Fig. 7. The parameter k of OFFDSR is varied as well as the number of timesteps to further illustrate the effect of the boundary conditions on the ranks.

APPENDIX F: CROSS-VALIDATION AND HYPERPARAMETER TUNING

We provide more technical details about the hyperparameter tuning used in the various algorithms and experiments. In all cases, we assume training and test sets have a chronological order, i.e., all matches in the train set happen earlier than those in the test set. Regardless of hyperparameters, all cross-validation folds provide the same exact train/test set to each algorithm for a fair comparison. Importantly, test sets are only used for evaluation.

All results displayed were computed with cross-validation which entailed using 50% of the total data as a train set and four time-steps as a test set. This interval was shifted by 1 time-step each fold. Figure 6 demonstrates this process. As a result of cross-validation, there are at most four different values for the same time-step. The reported results are an average of these values.

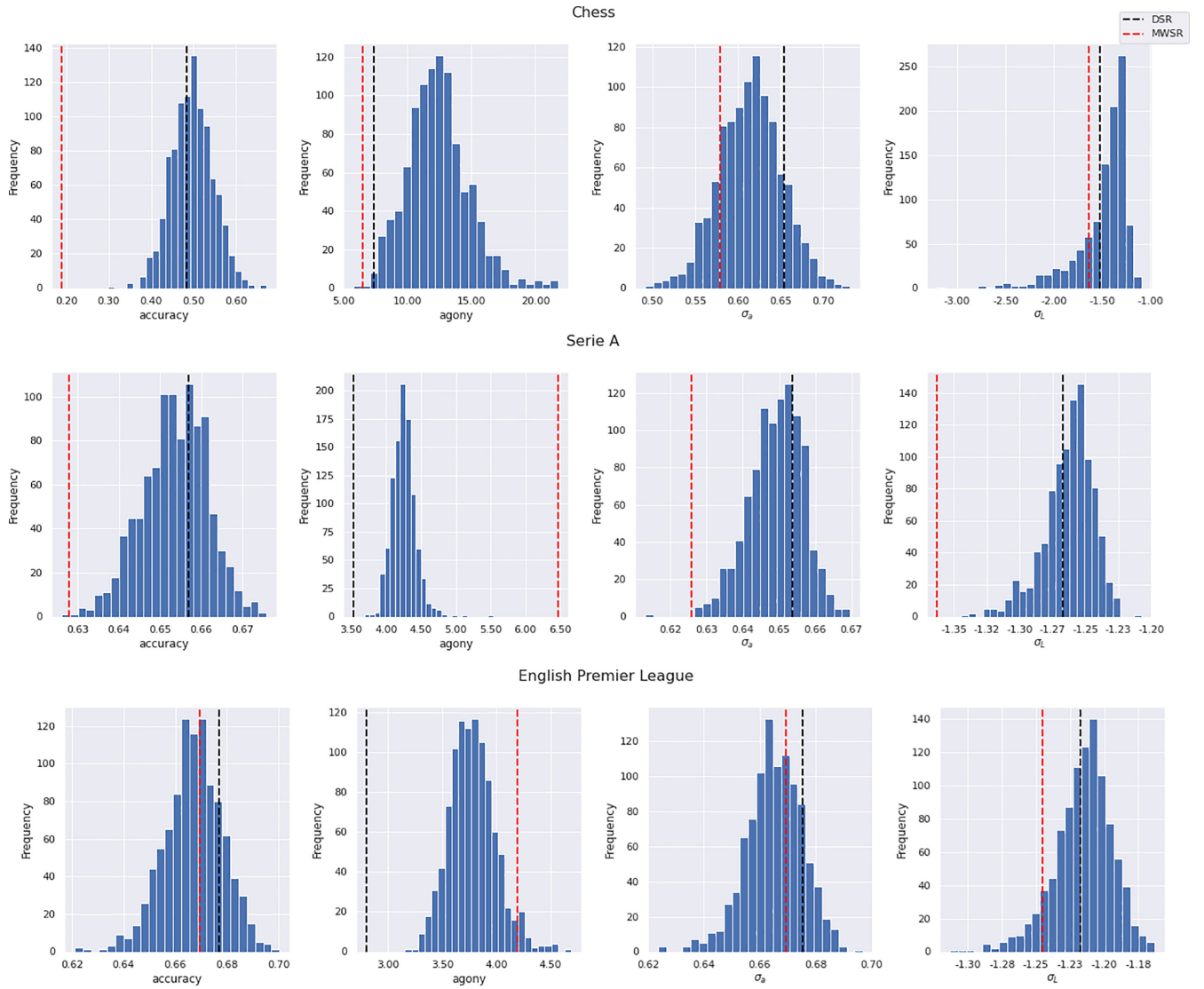

FIG. 9. Null model results on the chess, EPL and Serie A datasets. It is used to determine whether chronology is important. Each entry of the histogram is a different result of DSR on the aforementioned datasets, where time-steps have been randomly permutated; 1000 permutations were considered. The black and red dotted lines represent the results of DSR and mwSR, respectively, on the regular, chronologically ordered datasets.

As previously mentioned, we used grid-search to perform hyperparameter tuning. For Dynamical SpringRank, grid-search is divided into two steps: first, finding the order of magnitude of k and then progressively finding a more precise value. Refer to Algorithm 1 for the pseudocode of the procedure followed.

ALGORITHM I. Grid-Search.

1: $K = \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$
2: **for** $i \leftarrow -1, -2, -3$ **do**
2: **for** k in K **do**
4: Find k^*, the optimal k that produces best result
5: **end for**
6: Update interval $K = [k^* - 10^i, \quad k^* + 10^i]$
7: **end for**

Three algorithms (mwSR, TS, and WHR) require an optimal window size, τ_{opt}, for storing data in the training set. We chose this by varying the window size, calculating the average value for each performance metric inside the training set and then choosing the window size corresponding to the best of each of these values. Since the reported results are due to cross-validation, on average the window size of mwSR, TS, and WHR on the NBA dataset is $\tau_{\text{opt}} = 13, 23, 31$, respectively.

Next, Elo requires a scaling factor k which was determined through a grid-search in the interval [0.01,1). WL requires a decaying factor $\beta = 3$ and a weighting for indirect wins $\alpha = 0.005$, both of which were fixed with cross-validation. Finally, there are different versions of static SpringRank and we considered the standard version with regularization $\alpha = 0$.

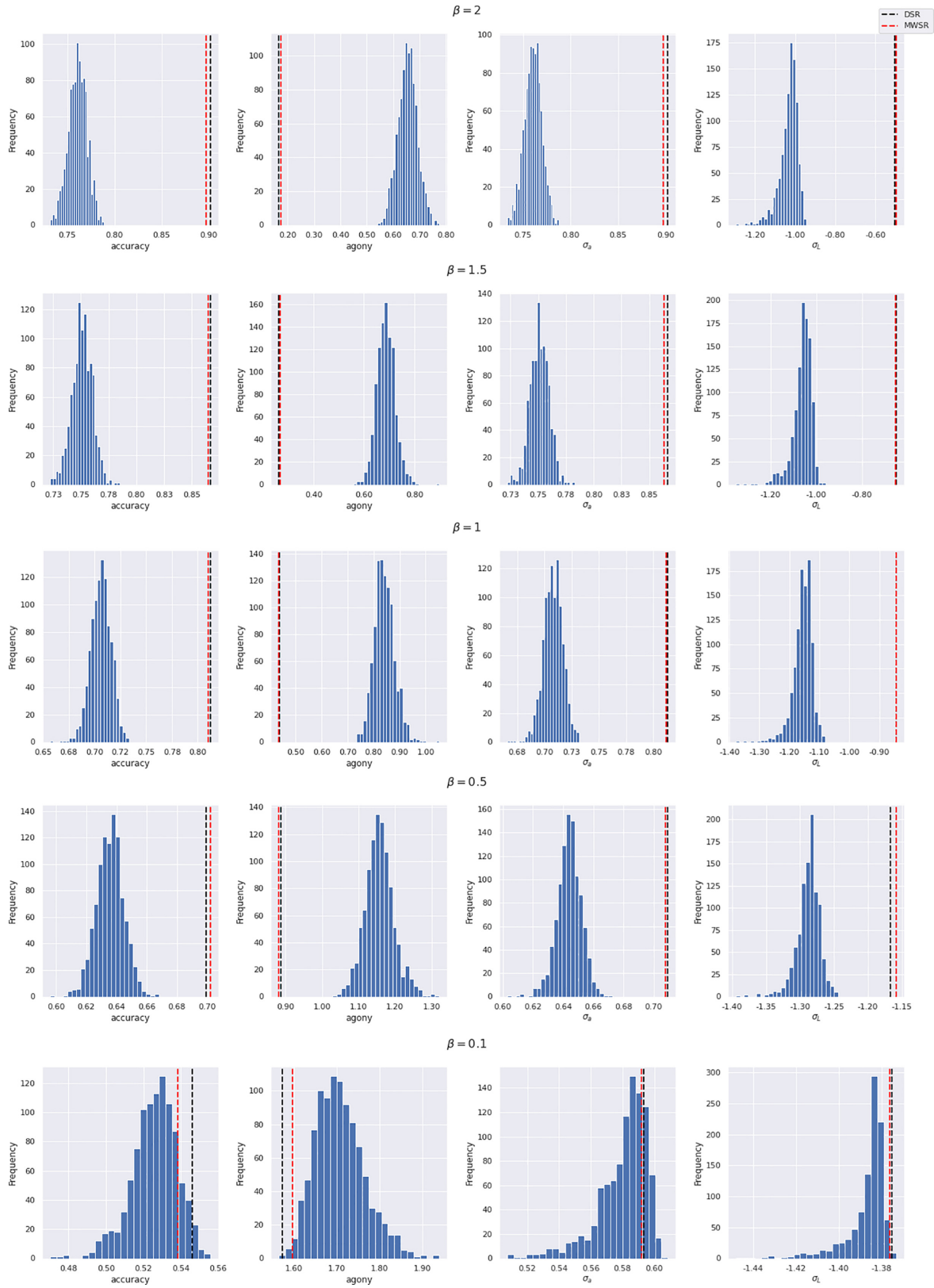

FIG. 10. Null model results on the synthetic dataset with varying levels of noise. It is used to determine whether chronology is important. Each entry of the histogram is a different result of DSR on the synthetic dataset where time-steps have been randomly permutated; 1000 permutations were considered. The black and red dotted lines represent the results of DSR and mwSR, respectively, on the regular, chronologically ordered dataset.

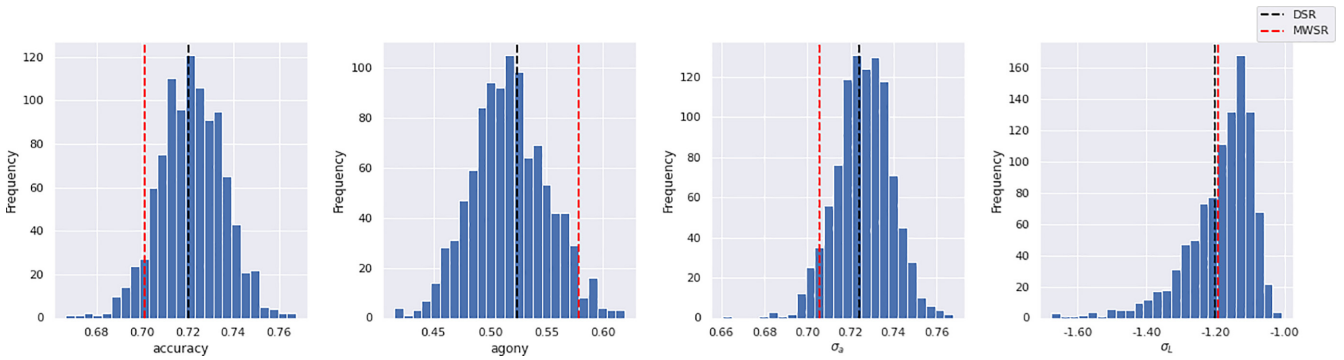

FIG. 11. Null model results on the synthetic dataset with static ranks. It is used to determine whether chronology is important. Each entry of the histogram is a different result of DSR on the synthetic dataset where time-steps have been randomly permutated; 1000 permutations were considered. The black and red dotted lines represent the results of DSR and mwSR, respectively, on the regular, chronologically ordered dataset.

APPENDIX G: IMPLEMENTATION OF OFFDSR AND ITS BOUNDARY CONDITIONS

In our experiments we created a realistic scenario in which each algorithm had to predict the future ranks of a node given its past interactions, as if the predictions were taking place in "real-time." Thus, only past information was given to each model. Offline Dynamical SpringRank depends on both past and future ranks [see Eq. (6)]. As a consequence of our experimental choice, the information it was given during the experiments was restricted to only the past. However, the description of OFFDSR in Sec. III relates more to a scenario where all information is available and ranks are inferred in hindsight.

Next, here we further discuss the boundary conditions of OFFDSR. Our choice for the boundary conditions on the ranks is implemented by removing the following terms from Eq. (6): $\mathbf{s}^{t-1}$ and $\mathbf{s}^{t+1}$for $t = 0$ and $t = T$, respectively. This is equivalent to $\mathbf{s}^{-1} = 0 = \mathbf{s}^{T+1}$. The effect of this boundary condition is that ranks close to the boundary conditions are slightly pulled towards zero. This has a greater influence on datasets with a small number of timesteps. However, the effect lessens with more timesteps. Alternate boundary conditions may be chosen, such as $\mathbf{s}^{-1} = \mathbf{s}^{0}$ and $\mathbf{s}^{T} = \mathbf{s}^{T+1}$. The effect of the two aforementioned boundary conditions on a toy example with only two nodes and one directed edge between them is illustrated in Figs. 7 and 8, respectively. We do not explore the effects of different boundary conditions in our experiments and leave it for future work.

APPENDIX H: SYNTHETIC EXPERIMENTS

1. Periodic evolution of synthetic ranks

We consider a periodic evolution of the ranks generated for synthetic experiments, expressed as Eq. (8). To add detail to the extraction process of the parameters, they were selected from a continuous uniform distribution. The interval of the distribution for parameters was as follows: $b_i, c_i \in [-1, 1)$, as it is the standard range for a cosine function; $\omega_i, \upsilon_i \in [-1, 2)$ to vary the frequency with which scores change, with larger increases reflected in values between 1 and 2 being less likely than values between -1 and 1; finally, $\phi_i \in [0, 1)$ to ensure that scores do not have the same rate of change at the beginning of the time interval.

2. Standard errors

We report standard errors on synthetic experiments where we vary the noise level represented by the parameter β in Table IV. These complement Table I in the main manuscript.

3. Results for varying network density

In Tables V and VI, we show results on synthetic data where we vary the network density represented by the parameter c.

4. Synthetic ranks in static scenarios

We consider static ranks $s_i^t = s_i$ generated synthetically using Eq. (8) as a sanity check of our permutation test for

TABLE VIII. Standard error of results from the synthetic data in a static framework.

Metric	Elo	OFFDSR	mwSR	DSR	SR	TS	W-L	WHR
Accuracy	0.0107	0.0109	0.0109	0.0109	0.0111	0.0101	0.0192	0.0099
Agony	0.0243	0.0247	0.0250	0.0261	0.0259	0.0242	0.0552	0.0242
σ_a	0.0059	0.0052	0.0099	0.0095	0.0085	0.0058	–	0.0056
σ_L	0.0423	0.0049	0.0380	0.0399	0.0223	0.0277	–	0.0251

TABLE IX. Standard error of results from real data

Dataset	Metric	Elo	OFFDSR	mwSR	DSR	SR	TS	WHR	W-L
NBA	Accuracy	0.0048	0.0049	0.0046	0.0048	0.0047	0.0050	0.0047	0.0049
	Agony	0.0587	0.0588	0.0621	0.0577	0.0529	0.0576	0.0553	0.0665
	σ_a	0.0021	0.0026	0.0045	0.0045	0.0040	0.0024	0.0022	–
	σ_L	0.0154	0.0029	0.0075	0.0063	0.0041	0.0097	0.0082	–
Chess	Accuracy	0.0213	0.0242	0.0246	0.0213	0.0214	0.0235	0.0226	0.0119
	Agony	1.2323	1.8357	1.6712	1.2301	1.2292	1.3637	1.3848	0.3262
	σ_a	0.0137	0.0134	0.0183	0.0179	0.0123	0.0144	0.0132	–
	σ_L	0.0773	0.0356	0.0672	0.0753	0.1090	0.0733	0.0526	–
EPL	Accuracy	0.0061	0.0067	0.0073	0.0058	0.0063	0.0064	0.0062	0.0079
	Agony	0.0920	0.1292	0.1429	0.0906	0.0935	0.1172	0.1082	0.1424
	σ_a	0.0026	0.0035	0.0068	0.0059	0.0060	0.0030	0.0032	–
	σ_L	0.0249	0.0038	0.0139	0.0145	0.0112	0.0154	0.0141	–
Serie A	Accuracy	0.0048	0.0051	0.0051	0.0049	0.0050	0.0050	0.0047	0.0059
	Agony	0.0881	0.1227	0.1282	0.0856	0.0930	0.1252	0.1174	0.1617
	σ_a	0.0020	0.0013	0.0047	0.0044	0.0045	0.0024	0.0022	–
	σ_L	0.0176	0.0013	0.0102	0.0104	0.0060	0.0107	0.0093	–

model selection between static and dynamic models. Results are shown in Tables VII and VIII.

APPENDIX I: REAL DATA EXPERIMENTS

1. Standard errors

We report the standard errors of the experiments on the real datasets in Table IX. These complement Table III in the main manuscript.

APPENDIX J: NULL MODEL EXPERIMENTS

1. Synthetic data

We report results of the null model experiments where we permute the chronological order of synthetic dynamic data in Fig. 10 and of synthetic static data in Fig. 11.

2. Real data

We report p values on the null model experiments on real data in Table X.

TABLE X. Null model p-value results on real data. Illustrated are the number of times (as a percentage) that the metric value on the randomized dataset is better than on the chronologically ordered dataset. Results are calculated over 1000 permutations.

Model	Metric	NBA	Chess	EPL	Serie A
DSR	Accuracy	0.0	0.594	0.183	0.359
	Agony	0.0	0.006	0.0	0.0
	σ_a	0.0	0.139	0.155	0.282
	σ_L	0.0	0.711	0.578	0.644
mwSR	Accuracy	0.0	1.0	0.411	0.999
	Agony	0.0	0.001	0.959	1.0
	σ_a	0.0	0.832	0.340	0.999
	σ_L	0.0	0.797	0.911	1.0

[1] C. De Bacco, D. B. Larremore, and C. Moore, Sci. Adv. **4**, eaar8260 (2018).
[2] P. Bonacich, Amer. J. Sociol. **92**, 1170 (1987).
[3] L. Page, S. Brin, R. Motwani, and T. Winograd, *The PageRank citation ranking: Bringing order to the web*, Tech. Rep. (Stanford InfoLab, Stanford, CA, 1999).
[4] S. Negahban, S. Oh, and D. Shah, Oper. Res. **65**, 266 (2017).
[5] I. Ali, W. D. Cook, and M. Kress, Manag. Sci. **32**, 660 (1986).
[6] P. Slater, Biometrika **48**, 303 (1961).
[7] M. Gupte, P. Shankar, J. Li, S. Muthukrishnan, and L. Iftode, in *Proceedings of the 20th International Conference on the World Wide Web* (ACM, New York, NY, 2011), pp. 557–566.
[8] E. Letizia, P. Barucca, and F. Lillo, PLoS One **13**, e0191604 (2018).
[9] F. Fogel, A. d'Aspremont, and M. Vojnovic, in *Advances in Neural Information Processing Systems* (MIT Press, Cambridge, MA, 2014), pp. 900–908.
[10] M. Cucuringu, IEEE Trans. Netw. Sci. Eng. **3**, 58 (2016).
[11] K. E. Train, *Discrete Choice Methods with Simulation* (Cambridge University Press, Cambridge, UK, 2009).
[12] R. A. Bradley and M. E. Terry, Biometrika **39**, 324 (1952).
[13] R. D. Luce, Psychological Review **66**, 81 (1959).
[14] R. J. Williams, A. Anandanadesan, and D. Purves, PLoS One **5**, e12092 (2010).
[15] R. J. Williams and D. W. Purves, Ecology **92**, 1849 (2011).
[16] A. Z. Jacobs, J. A. Dunne, C. Moore, and A. Clauset, arXiv:1505.04741.
[17] B. Ball and M. E. Newman, Netw. Sci. **1**, 16 (2013).
[18] L. Iacovissi and C. De Bacco, Sci. Rep. **12**, 8992 (2022).

[19] P. D. Hoff, A. E. Raftery, and M. S. Handcock, J. Am. Stat. Assoc. **97**, 1090 (2002).
[20] G. T. Cantwell and C. Moore, Phys. Rev. E **105**, L052303 (2022).
[21] A. E. Elo, *The Rating of Chessplayers, Past and Present* (Arco Publishing, New York, NY, 1978).
[22] M. E. Glickman, Boston University **16**, 9 (1995).
[23] E. Araya, E. Karlé, and H. Tyagi, Information and Inference A Journal of the IMA **12**, 2224 (2023).
[24] J. Park and M. E. Newman, J. Stat. Mech.: Theory Exp. (2005) P10014.
[25] S. Motegi and N. Masuda, Sci. Rep. **2**, 904 (2012).
[26] R. Herbrich, T. Minka, and T. Graepel, in *Advances in Neural Information Processing Systems* (MIT Press, Cambridge, MA, 2007), pp. 569–576.
[27] P. Dangauthier, R. Herbrich, T. Minka, and T. Graepel, in *Advances in Neural Information Processing Systems* (MIT Press, Cambridge, MA, 2008), pp. 337–344.
[28] R. Coulom, in *Proceedings of the International Conference on Computers and Games* (Springer, Berlin, 2008), pp. 113–124.
[29] R. Peng and S. Vempala, in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)* (SIAM, Philadelphia, PA, 2021), pp. 504–521.
[30] https://github.com/cdebacco/DynSpringRank.
[31] J. C. Flack, M. Girvan, F. de Waal, and D. C. Krakauer, Nature (London) **439**, 426 (2006).
[32] M. E. Glickman, *Paired Comparison Models with Time-varying Parameters* (Harvard University, Cambridge, MA, 1993).