BIOINFORMATICS

Supplementary Information for:

Exploring the structure and function of temporal networks with dynamic graphlets

Hulovatyy, Y., Chen, H., and Milenković T.*

Department of Computer Science and Engineering, Interdisciplinary Center for Network Science and Applications, and ECK Institute for Global Health, University of Notre Dame, IN 46556, USA

S1 COMPUTING THE NUMBER OF DYNAMIC GRAPHLET TYPES OF A GIVEN SIZE

Here, we expand the discussion from the main paper on how to compute $D(n,k)$, the number of dynamic graphlet types with n nodes and k events.

Since at least $n-1$ edges are needed to connect n nodes, $D(n,k)=0$ for $k<n-1$. Moreover, since our events are undirected, $D(2,k)=1$, for any k. To compute $D(n,k)$ when $n\geq 3$ and $k\geq n-1$, notice that each dynamic graphlet with k events can be formed from a dynamic graphlet with $k-1$ events and either $n-1$ or n nodes, by adding a new event between some two existing nodes or between an existing node and a new node, respectively.

In the first case, we take a dynamic graphlet with n nodes and $k-1$ events and add a new event between its existing nodes, in order to obtain a dynamic graphlet with n nodes and k events (e.g., construct D_6 from D_2). Due to the Δt-connectivity constraint, this new event has to involve at least one of the two nodes participating in event $(k-1)$. We can add the new event in $2n-3$ different ways: between one of these nodes and the "remaining" $n-2$ nodes (which is $2(n-2)=2n-4$ ways) or just duplicate event $(k-1)$.

In the second case, we take a dynamic graphlet with $n-1$ nodes and $k-1$ events and add an event from one of its nodes to the new (n^{th}) node, in order to obtain a dynamic graphlet with n nodes and k events (e.g., construct D_4 from D_1). For $n-1\geq 3$, there are two ways to do this, since there are two potential candidates for this new event (the two endpoints of event $(k-1)$). Note that since events are undirected, for $n-1=2$, the two nodes are indistinguishable from our point of view, and so we have only one way to construct a new dynamic graphlet with 3 nodes and k events.

In summary, we can get $2n-3$ new dynamic graphlets with n nodes and k events from each dynamic graphlet with n nodes and $k-1$ events. Moreover, we can get two new dynamic graphlets with n nodes and k events from each dynamic graphlet with $n-1$ nodes and $k-1$ events. The only exception is for $n=3$, since we can get only one new dynamic graphlet with three nodes from a dynamic graphlet with two nodes, as these two nodes are indistinguishable. Importantly, since each dynamic graphlet with k events has a unique $(k-1)$-"prefix" from which it was extended, all of these new dynamic graphlets with n nodes will be different. Thus, we get the following recursive formulas for $D(n,k)$: $D(3,k)=3D(3,k-1)+D(2,k-1), n=3$; and $D(n,k)=(2n-3)D(n,k-1)+2D(n-1,k-1), n>3$. By expanding the formulas for few smallest values of n and k, we get the following closed-form solution (Supplementary Table S2):

$$D(n,k)=\sum_{i=0}^{n-2}\frac{(-1)^{n+i}\binom{n-2}{i}(2i+1)^{k-1}}{2(n-2)!}, n\geq 3.$$

S2 CONSTRAINED COUNTING OF DYNAMIC GRAPHLETS IN A NETWORK

Here, we expand the discussion from the main paper on the constrained dynamic graphlet counting procedure.

A network having dense neighborhoods with many events between the same nodes will have large counts of dynamic graphlet types. This is because for a given dynamic graphlet instance, there will be many Δt-adjacent candidate events which can be used to "grow" this graphlet. For each of these possible graphlet extensions, we will again have many possibilities for further extension, and so on. For example, consider a snapshot-based network representation where each snapshot is the same dense graph. A large number of different dynamic graphlet instances will be detected, yet many of them will just be artifacts of the consecutive snapshots "sharing" the dense network structure. As an (rather extreme) example, consider all snapshots being the same fully connected graph. In this case, for a dynamic graphlet instances ending with some event, we will have multiple events that can be used to grow this graphlet, resulting in large counts for all possible dynamic graphlet

*To whom correspondence should be addressed

types. Clearly, considering all of these possible extensions will be computationally expensive; moreover, not all of the detected dynamic graphlet instances will have meaningful interpretations.

To address this and remove the likely redundant graphlet counts, which is expected to reduce computational complexity of dynamic graphlet counting, we propose a modification to the counting process, as follows. When we are extending a dynamic graphlet ending with event $e_1 = (u_1, v_1, t_1, \sigma_1)$ with a new event $e_2 = (u_2, v_2, t_2, \sigma_2)$, if $\{u_1, v_1\} = \{u_2, v_2\}$, i.e., if the two events correspond to the same static edge, then we impose the same two conditions as in the regular counting procedure from Section 2.2 in the main paper: 1) the two events e_1 and e_2 must be Δt-adjacent with $t_2 > t_1$, and 2) the two events must share a node. Otherwise, if $\{u_1, v_1\} \neq \{u_2, v_2\}$, i.e., if the two events correspond to two different static edges, we also add a new third condition: to extend the dynamic graphlet ending with event e_1 with event e_2, u_2 and v_2 cannot interact between the starting times of e_1 and e_2 (i.e., $\nexists e' = (u_2, v_2, t', \sigma') \in E$ with $t_1 - \sigma' \leq t' < t_2$).

Intuitively, the new third condition requires a "causal" relationship between e_1 and e_2: u_2 and v_2 start their interaction only after the end of e_1 (note that there could still be an event involving u_2 and v_2 sometime before the start of e_1). That is, in order to extend a dynamic graphlet ending with event e_1 with some event e_2, the two nodes participating in e_2 should not interact with each other between the start of e_1 and the start of e_2, unless e_1 and e_2 involve the same nodes (otherwise, the counting process is as in the regular dynamic graphlet procedure). This reduces the number of likely redundant temporal subgraphs that are being considered, which in turn reduces the running time.

We split the counting procedure into the two cases (e_1 and e_2 being the same static edge vs. different static edges) as we want to impose the new third condition only in the latter case. This is because in the former case we still want to count dynamic graphlet types having consecutive repetitions of the same event, e.g., D_1 or D_4. And if we imposed the third condition in the former case as well, such a dynamic graphlet type would never be counted.

We refer to this modified counting procedure as *constrained dynamic graphlet counting*. Note that the only difference in case of constrained dynamic graphlet counting compared to the original counting procedure is which candidate events are chosen to extend a given dynamic graphlet (as determined by procedure *GetNextEvents* in Supplementary Algorithm S3). In the main paper, we illustrate the distinction between regular and constrained dynamic graphlet counting procedures. Clearly, constrained dynamic graphlet counting allows for examining fewer instances of a given dynamic graphlet type compared to regular dynamic graphlet counting, because the former excludes from consideration graphlet instances that are likely artifacts of repeated events, unlike the latter. As a consequence, constrained dynamic graphlet counting is expected to be more computationally efficient in terms of running time.

S3 EXPERIMENTAL SETUP

Graphlet methods under consideration and network construction. When generating an aggregate or snapshot-based representation of the given temporal network, we began our analysis by testing in detail w values of 1, 2, 3, 5, and 10 on one of our data sets. Since we observed no qualitative differences in results produced by the different choices of this parameter, we continued with the choice of $w = 1$, and we report the corresponding results throughout the main paper. We also tested multiple values for t_w in each data set, and again we saw no significant qualitative differences in the results. Hence, unless noted otherwise, throughout the paper, we report results for $t_w = 2$ (the unit of time for this parameter depends on the data set).

Network classification. The first network evolution model that we use in the context of this analysis was designed to simulate evolution of real-world (social) networks, and it incorporates the following parameters: node arrival rate, initiation of an edge by a node, and selection of edge destination. Specifically, the model is parameterized by the node arrival function $N(t)$ that corresponds to the number of nodes in the network at a given time, parameter λ that controls the lifetime of a node, and parameters α and β that control how active the nodes are in adding new edges. By choosing different options for the model parameters, we can generate networks with different evolution processes. In particular, for our analysis, we test three different types of the node arrival function (linear, quadratic, and exponential) and two sets of parameters corresponding to edge initiation ($\lambda_1 = 0.032$, $\alpha_1 = 0.8$, $\beta_1 = 0.002$, and $\lambda_2 = 0.02$, $\alpha_2 = 0.9$, $\beta_2 = 0.004$), resulting in six different network classes. We also test a modification of the network evolution model, in which each node upon arrival simply adds a fixed number of edges (in our case, 20) according to preferential attachment and then stops. Intuitively, this modification corresponds to preferential attachment model extended with a node arrival function. In this way, we create three additional network classes, one for each of the three node arrival functions, resulting in nine different network classes in total.

In addition to the above network model originating from social network domain, we perform an equivalent analysis using biological network models. Specifically, we use the following models: geometric gene duplication model with probability cutoff (GEO-GD) and scale-free gene duplication model (SF-GD). GEO-GD starts with a small initial seed network and then iteratively adds new nodes by choosing as the parent an existing node uniformly at random, and by placing a child node with probability p at a randomly chosen distance of at most ϵ from the parent and with probability $1 - p$ at a randomly chosen distance of at most 10ϵ from the parent. Here, ϵ is the same parameter as in the definition of a geometric random graph. Similarly, SF-GD also starts from an initial seed network and then grows it by adding new nodes while relying on principles of gene duplication and mutation. This model has two parameters, p and q, corresponding to the probability of the child node keeping interaction with the parent and the probability of the child node forming new connections, respectively. We distinguish between four variations of these two models: GEO-GD with $p = 0.3$, GEO-GD with $p = 0.7$, SF-GD with $p = 0.3$ and $q = 0.7$, and SF-GD with $p = 0.7$ and $q = 0.6$, as these parameter combinations accurately mimic real-world networks. In order to grow the GEO-GD and SF-GD model networks non-uniformly, as in the case with the above social network model, we add a node arrival function on top of GEO-GD and SF-GD. Specifically, we use linear node arrival function for GEO-GD and exponential node arrival function for SF-GD. Note that we intentionally use different node arrival functions for the different biological network models, just as we did with the different classes of the

social network model. As with the social network model, we form snapshot-based network representations, test various network sizes (from 1000 to 3000 nodes), and construct 25 networks for each class (Section 2.3 in the main paper). Note that here, when we use snapshot-based representation, we report results for $t_w = 5$. We then evaluate graphlet-based methods in the same way as for social network models. To account for at least 90% of variance, here we also need to keep only the first two PCA components.

Node classification. For the network used in this analysis, we report results for the following network construction parameters: $w = 1$ and $t_w = 2$ months. Note that we tested other parameter values as well ($w = 2, 3, 5, 10$; $t_w = 1$ week, $t_w = 2$ weeks, $t_w = 1$ month, and $t_w = 3$ months), and all results were qualitatively similar.

Evaluation strategy. Here, we expand our discussion from the main paper on how we measure the PCA performance of a given graphlet approach.

First, we take all possible *pairs* of objects and retrieve them in the order of increasing distance, starting from the closest ones. We retrieve the object pairs in increments of $k\%$ (including ties), where we vary k from 0% to 100% in increments of 0.01% until we retrieve top 1% of all pairs and in increments of 1% afterwards. If we retrieve a pair with two objects of the same ground truth class, the pair is a true positive, otherwise the pair is a false positive. At a given step, for all pairs that we do not retrieve, the given pair is either a true negative (if it contains objects of different classes) or a false negative (if it contain objects of the same class). Then, at each value of k, we compute precision, the fraction of correctly retrieved pairs out of all retrieved pairs, and recall, the fraction of correctly retrieved pairs out of all correct pairs. We find the value of k where precision and recall are equal, and we refer to the resulting precision and recall value as the break-even point. Since lower precision means higher recall, and vice versa, we summarize the two measures into F-score, their harmonic mean, and we report the maximum F-score over all values of k. To summarize these results over the whole range of k, we measure average method accuracy by computing the area under the precision-recall curve (AUPR). Moreover, we compute an alternative classification accuracy measure, namely the area under the receiver operator characteristic curve (AUROC), which corresponds to the probability of a method ranking a randomly chosen positive pair higher than a randomly chosen negative pair (and so the AUROC value of 0.5 corresponds to a random result). AUPRs are considered to be more credible than AUROCs when there exists imbalance between the size of the set of object pairs that share a class and the size of the set of object pairs that do not share a class.

Second, we split all pairs of objects (i.e., their graphlet-based PCA distances) into two classes: correct pairs (each containing two objects of the same class) and incorrect pairs (each containing two objects of two different classes). Then, we compare distances between correct and incorrect pairs, expecting that distances of the correct pairs would be statistically significantly lower than distances of the incorrect pairs. Here, we compare two sets of distances via Wilcoxon rank-sum test.

Finally, for each of the above evaluation tests, we evaluate all graphlet methods against a random approach. First, as a simple random approach (which favors the graphlet methods the most), we randomly embed objects into Euclidean space, compute the objects' pairwise Euclidean distances, and evaluate the resulting random approach in the same way as above. Second, as a more restrictive random approach (which favors the graphlet methods the least), for each graphlet method, we keep its actual PCA distances between objects, and we just randomly permute the object classes/labels before we evaluate the results. For each randomization approach, we compute its results as an average over 10 different runs. We report as "random" approach's results the highest-scoring values over all of the randomization schemes, to gain as much confidence as possible into the graphlet approaches' results.

S4 EFFECT OF GRAPHLET SIZE ON RESULTS

Number of graphlet nodes. We test the effect of graphlet size in terms of the number of nodes on the result quality, for all graphlet methods. For network classification, increasing the number of graphlet nodes surprisingly leads to inferior accuracy, for all graphlet methods (Supplementary Table S3). On the other hand, in node classification, for static and static-temporal graphlets, accuracy is similar for all graphlet sizes, with 4-node graphlets showing marginal superiority, while for dynamic and constrained dynamic graphlets, larger number of nodes improves accuracy (Supplementary Table S5). In terms of the running time, as expected, larger number of nodes increases computational complexity in all cases.

Note that with node classification, unlike with network classification, the best parameter version of constrained dynamic graphlet counting is more accurate than the best parameter version of regular dynamic graphlet counting. We note, however, that due to the differences in the counting process, constrained dynamic graphlet counting allows us to consider larger graphlet sizes (e.g., six or seven nodes) that are not attainable when using regular dynamic graphlet counting due to computational constraints (Supplementary Table S5). And it is at these large graphlet sizes of six or seven nodes where constrained dynamic graphlet counting perform the best. So, in order to evaluate which one is more accurate, regular or constrained dynamic graphlet counting, it might not be fair to compare the two methods' best parameter versions, due to differences in the considered graphlet sizes. Nonetheless, even if we compare regular or constrained dynamic graphlet counting with the same parameters, we find that constrained dynamic graphlet counting still demonstrates better results (Supplementary Table S5).

Number of graphlet events. Also, we test the effect of graphlet size in terms of the number of events on the result quality, for dynamic and constrained dynamic graphlets (static and static-temporal graphlets do not deal with events, i.e., temporal edges). For network classification, the number of events does not affect the accuracy (Supplementary Table S3). On the other hand, in node classification, for a fixed number of nodes, the increase in the number of events marginally increases accuracy for dynamic graphlets but decreases accuracy for constrained dynamic graphlets (Supplementary Table S5). In terms of the running time, larger number of events increases computational complexity, although the level of running time increase is less pronounced than when increasing the number of nodes.

A possible explanation why increasing graphlet size does not always improve accuracy. Here, we provide a possible explanation behind our observation that increasing graphlet size sometimes leads to decrease in result accuracy in the task of network classification but not in the task of node classification. Namely, in network classification, we use synthetic networks generated from a *random* graph model, while in node classification we use *real-world* networks. A random graph model might not be able to produce large and dense graphlet structures. Now, given two synthetic networks of different classes, whereas their comparison via smaller graphlets only could correctly identify the networks as dissimilar, their comparison via larger graphlets (with zero counts) as well could mistakenly identify the networks as similar. This is because there are many more larger than smaller graphlets, and thus, the many zero counts for the larger graphlets would match between the networks, wrongly indicating their similarity. However, note that our PCA framework, which reduces graphlet count vectors to a lower dimension (Section 2.3 in the main paper), should remove the effect of zero graphlet counts, because it should be preserving only the most relevant graphlet-based information about the network of interest.

An alternative reason why increasing graphlet size (and the number of graphlet nodes in particular) would lead to decrease in performance is as follows. Some graphlet types, such as a claw (e.g., G_1 or G_4 in Fig. 1 in the main paper), typically result in an order of magnitude larger counts than other graphlet types, simply because of presence of high-degree (hub) nodes in real-world networks or any scale-free (model) networks. This difference in the magnitude of counts for such highly frequent graphlet types compared to counts of less frequent graphlet types becomes more pronounced as the graphlet size increases. And such extremely dominating graphlet counts could potentially confuse our PCA framework and mistakenly identify the highly frequent graphlet types as more relevant than the less frequent graphlet types. For this reason, one could "rescale" all graphlet counts to get them to the same (or similar) order of magnitude, by, e.g., taking a logarithm of each count.[1]

Clearly, determining why increase in graphlet size does not necessarily improve the accuracy of results requires further theoretical and empirical analyses, and this is subject of future work.

[1] Pržulj, N., Corneil, D. G. and Jurisica, I. (2004) Modeling interactome: scale-free or geometric? *Bioinformatics*, 20, 3508-3515.

Supplementary Algorithm S1 Enumerate all dynamic graphlet types with up to n_{max} nodes and k_{max} events

procedure ENUMERATE(n_{max},k_{max})
  $k \leftarrow 2$
  $i \leftarrow 1$
  $G_0 \leftarrow$'12'
  $Graphlets[1] \leftarrow \{G_0\}$ ▷ set of dynamic graphlets with a given number of events
  while $k \leq k_{max}$ **do**
    $Graphlets[k] \leftarrow \emptyset$
    for all $G \in Graphlets[k-1]$ **do**
      if $|V(G)| \leq n_{max}$ **then**
        $s \leftarrow G[2k-4]$ ▷ recall that G contains $2k-2$ numbers
        $t \leftarrow G[2k-3]$
        $u \leftarrow \max(G)+1$ ▷ u is the "new node"
        if $|V(G)| = 2$ **then** ▷ $V(G)$ is just the set of different numbers in G
          if $|V(G)| < n_{max}$ **then**
            $G_i \leftarrow G + [s,t]$
            $Graphlets[k] \leftarrow Graphlets[k] \cup \{G_i\}$
            $i \leftarrow i+1$
          end if
          $G_i \leftarrow G + [t,u]$
          $Graphlets[k] \leftarrow Graphlets[k] \cup \{G_i\}$
          $i \leftarrow i+1$
        else
          for all $v \in V(G) - \{s\}$ **do**
            $G_i \leftarrow G + [min(s,v), max(s,v)]$ ▷ for consistency, keep each event in increasing order
            $Graphlets[k] \leftarrow Graphlets[k] \cup G_i$
            $i \leftarrow i+1$
          end for
          for all $v \in V(G) - \{s,t\}$ **do**
            $G_i \leftarrow G + [min(t,v), max(t,v)]$
            $Graphlets[k] \leftarrow Graphlets[k] \cup G_i$
            $i \leftarrow i+1$
          end for
          if $|V(G)| < n_{max}$ **then**
            $G_i \leftarrow G + [s,u]$
            $G_{i+1} \leftarrow G + [t,u]$
            $Graphlets[k] \leftarrow Graphlets[k] \cup \{G_i, G_{i+1}\}$
            $i \leftarrow i+2$
          end if
        end if
      end if
    end for
    $k \leftarrow k+1$
  end while
end procedure

Supplementary Algorithm S2 Count in the input network the frequency of each of dynamic graphlets with up to n_{max} nodes and k_{max} events

procedure GRAPHLETCOUNT($G, n_{max}, k_{max}, \Delta t$)
  $Counts \leftarrow dict()$
  $NCounts \leftarrow dict()$
  for all $e \in G.Events()$ **do** GRAPHLETCOUNTFROMPREFIX($G, [e], Counts, NCounts, n_{max}, k_{max}, \Delta t$)
  end for
end procedure

Supplementary Algorithm S3 Count dynamic graphlets with up to n_{max} nodes and k_{max} events starting from a given prefix

procedure GRAPHLETCOUNTFROMPREFIX($G, prefixG, Counts, NCounts, n_{max}, k_{max}, \Delta t$)
  $prefEncoding, prefNodeCodes \leftarrow$ ENCODEGRAPHLET($prefixG$)
  $Counts[prefixEncoding] \leftarrow Counts[prefixEncoding] + 1$
  $prefNodes \leftarrow prefNodeCodes.keys()$
  for all $v \in prefNodes$ **do**
    if $|prefNodes| = 2$ **then**
      $nodeCode \leftarrow$'1'
    else
      $nodeCode \leftarrow prefNodeCodes[v]$
    end if
    $NCounts[v][prefixEncoding+\text{'\_'}+nodeCode] \leftarrow NCounts[v][prefixEncoding+\text{'\_'}+nodeCode] + 1$
  end for
  if $|prefixG| < k_{max}$ **then**
    for all $e \in$ GETNEXTEVENTS($G, prefixG[|prefixG| - 1], \Delta t$) **do** ▷ GetNextEvents returns all events that share a node with a given event and occur within Δt after it
      if $|prefNodes| < n_{max}$ or $e.EndPoints() \subseteq prefNodes$ **then**
        GRAPHLETCOUNTFROMPREFIX($G, prefixG + [e], Counts, NCounts, n_{max}, k_{max}, \Delta t$)
      end if
    end for
  end if
end procedure

Supplementary Algorithm S4 Encode a dynamic graphlet

procedure ENCODEGRAPHLET($Graphlet$)
  $encoding \leftarrow$''
  $nodeCodes \leftarrow dict()$
  $i \leftarrow 1$
  for all $e \in Graphlet$ **do**
    $u, v \leftarrow e.EndPoints()$
    $updated \leftarrow False$
    if $!nodeCodes.hasKey(u)$ **then**
      $nodeCodes[u] \leftarrow i$
      $i \leftarrow i + 1$
      $updated \leftarrow True$
    end if
    if $!nodeCodes.hasKey(v)$ **then**
      $nodeCodes[v] \leftarrow i$
      $i \leftarrow i + 1$
      $updated \leftarrow True$
    end if
    if $i = 4$ and $updated = True$ **then**▷ for consistency, replace '121212 . . . 1213' with '121212 . . . 1223' - need to swap codes of '1' and '2'
      if ($nodeCodes[u] = 1$ and $nodeCodes[v] = 3$) or ($nodeCodes[u] = 3$ and $nodeCodes[v] = 1$) **then**
        $nodeCodes[nodeCodes.GetKey(1)] \leftarrow 2$
        $nodeCodes[nodeCodes.GetKey(2)] \leftarrow 1$
      end if
    end if
    $encoding \leftarrow encoding + \min(nodeCodes[u], nodeCodes[v]) + \max(nodeCodes[u], nodeCodes[v])$
  end for
  return $encoding, nodeCodes$
end procedure

	1	2	3	4	5	6	7	8	9	10
G_0	1	1	1	1	1	1	1	1	1	1
G_1	0	1	3	7	15	31	63	127	255	511
G_2	0	0	1	6	25	90	301	966	3,025	9,330
G_3	0	0	1	5	18	56	161	441	1,170	3,036
G_4	0	0	1	6	25	90	301	966	3,025	9,330
G_5	0	0	0	1	7	34	140	525	1,855	6,294
G_6	0	0	0	6	54	320	1,576	7,006	29,238	117,048
G_7	0	0	0	0	12	150	1,190	7,658	43,690	230,760
G_8	0	0	0	0	0	10	174	1,842	15,398	111,972
G_9	0	0	0	1	6	26	97	332	1,074	3,339
G_{10}	0	0	0	2	17	94	429	1,758	6,735	24,668
G_{11}	0	0	0	1	10	65	350	1,701	7,770	34,105
G_{12}	0	0	0	0	10	121	926	5,731	31,359	158,475
G_{13}	0	0	0	0	4	42	286	1,598	7,968	36,928
G_{14}	0	0	0	0	9	120	1,001	6,716	39,747	217,106
G_{15}	0	0	0	0	1	8	43	193	781	2,955
G_{16}	0	0	0	0	7	70	456	2,440	11,649	51,648
G_{17}	0	0	0	0	0	62	1,037	10,640	86,462	611,904
G_{18}	0	0	0	0	0	7	109	1,055	8,148	55,084
G_{19}	0	0	0	0	0	34	532	5,104	38,846	257,884
G_{20}	0	0	0	0	0	5	68	573	3,852	22,660
G_{21}	0	0	0	0	0	26	354	3,010	20,522	122,906
G_{22}	0	0	0	0	0	0	50	1,063	13,549	134,524
G_{23}	0	0	0	0	0	0	90	1,954	25,198	251,952
G_{24}	0	0	0	0	0	0	183	3,676	44,367	418,109
G_{25}	0	0	0	0	0	0	69	1,287	14,382	125,419
G_{26}	0	0	0	0	0	0	0	594	15,442	235,160
G_{27}	0	0	0	0	0	0	0	219	5,400	77,959
G_{28}	0	0	0	0	0	0	0	0	1,133	35,859
G_{29}	0	0	0	0	0	0	0	0	0	796

Table S1. The number of dynamic graphlets with a given number of events (columns) that have as their backbone the same static graphlet (rows), for all static graphlets with up to five nodes.

	1	2	3	4	5	6	7	8	9	10
2	1	1	1	1	1	1	1	1	1	1
3	0	1	4	13	40	121	364	1,093	3,280	9,841
4	0	0	2	18	116	660	3,542	18,438	94,376	478,440
5	0	0	0	4	64	680	6,080	49,644	384,384	2,879,440
6	0	0	0	0	8	200	3,160	40,600	464,688	4,950,960
7	0	0	0	0	0	16	576	12,656	220,416	3,353,952
8	0	0	0	0	0	0	32	1,568	45,696	1,034,880
9	0	0	0	0	0	0	0	64	4,096	152,832
10	0	0	0	0	0	0	0	0	128	10,368
11	0	0	0	0	0	0	0	0	0	256
	1	2	7	36	229	1,678	13,755	124,064	1,217,065	12,870,970

Table S2. The number of dynamic graphlet types with up to ten nodes (rows) and ten events (columns). The last row shows cumulative results over all the preceding rows.

Method	AUPR	AUROC	Break-even point	Maximum F-score	Running time, s
Static, 3-node	0.507	0.935	0.508	0.613	3.3 (1.785)
Static, 4-node	0.423	0.882	0.463	0.468	3.3 (1.785)
Static, 5-node	0.321	0.807	0.341	0.376	3.3 (1.785)
Static-temporal, 3-node	0.784	0.947	0.702	0.707	3.7 (0.173)
Static-temporal, 4-node	0.498	0.826	0.475	0.476	3.7 (0.173)
Static-temporal, 5-node	0.374	0.790	0.379	0.390	3.7 (0.173)
Dynamic, 3-event, 3-node	**0.960**	**0.994**	**0.884**	**0.885**	**0.6 (0.116)**
Dynamic, 5-event, 3-node	**0.960**	**0.994**	**0.884**	**0.885**	0.7 (0.104)
Dynamic, 7-event, 3-node	**0.960**	**0.994**	**0.884**	**0.885**	1.4 (0.149)
Dynamic, 6-event, 4-node	0.714	0.937	0.656	0.660	4.8 (0.875)
Constrained dynamic, 3-event, 3-node	0.949	0.993	0.881	0.881	**0.6 (0.188)**
Constrained dynamic, 5-event, 3-node	0.949	0.993	0.881	0.881	0.7 (0.145)
Constrained dynamic, 7-event, 3-node	0.949	0.993	0.881	0.881	1.5 (0.206)
Constrained dynamic, 6-event, 4-node	0.740	0.939	0.672	0.675	4.2 (0.684)
Random	0.107 (0.002)	0.499 (0.005)	0.108 (0.006)	0.194 (0.000)	-

Table S3. Detailed network classification results for the different methods and different parameters in each method. Different columns correspond to different performance measures. In a given column, the value in bold corresponds to the best result over all methods. Numbers in parentheses correspond to standard deviations. For illustration purposes, graphlet counting running times are shown for one of the nine network classes (using the exponential node addition function and the first set of edge initiation parameters (Supplementary Section S3)); running times for the remaining network classes are shown in Supplementary Table S4. Note that for static and static-temporal graphlets, running times for 3- and 4-node graphlets are the same as for 5-node graphlets simply because their implementations compute graphlet counts for all up to 5-node graphlets by default and then they compute graphlet counts for smaller graphlet sizes simply by removing counts corresponding to the larger graphlet sizes.

	Linear			Quadratic			Exponential		
Method	Type 1	Type 2	PA	Type 1	Type 2	PA	Type 1	Type 2	PA
Static, 3-node	5.4 (2.914)	18.1 (4.957)	1316.2 (131.210)	4.0 (1.685)	10.1 (2.399)	1301.6 (131.622)	3.3 (1.785)	3.4 (1.628)	1195.2 (11.294)
Static, 4-node	5.4 (2.914)	18.1 (4.957)	1316.2 (131.210)	4.0 (1.685)	10.1 (2.399)	1301.6 (131.622)	3.3 (1.785)	3.4 (1.628)	1195.2 (11.294)
Static, 5-node	5.4 (2.914)	18.1 (4.957)	1316.2 (131.210)	4.0 (1.685)	10.1 (2.399)	1301.6 (131.622)	3.3 (1.785)	3.4 (1.628)	1195.2 (11.294)
Static-temporal, 3-node	47.7 (0.710)	49.1 (0.847)	48.5 (1.011)	23.9 (0.593)	24.2 (0.603)	24.8 (0.318)	3.7 (0.173)	3.7 (0.176)	2.8 (0.107)
Static-temporal, 4-node	47.7 (0.710)	49.1 (0.847)	48.5 (1.011)	23.9 (0.593)	24.2 (0.603)	24.8 (0.318)	3.7 (0.173)	3.7 (0.176)	2.8 (0.107)
Static-temporal, 5-node	47.7 (0.710)	49.1 (0.847)	48.5 (1.011)	23.9 (0.593)	24.2 (0.603)	24.8 (0.318)	3.7 (0.173)	3.7 (0.176)	2.8 (0.107)
Dynamic, 3-event, 3-node	0.8 (0.153)	1.2 (0.317)	2.4 (0.190)	0.6 (0.143)	0.9 (0.130)	2.9 (0.333)	0.6 (0.116)	0.8 (0.208)	8.0 (0.486)
Dynamic, 5-event, 3-node	0.9 (0.288)	1.4 (0.297)	2.5 (0.365)	0.8 (0.241)	1.1 (0.181)	2.9 (0.263)	0.7 (0.104)	0.9 (0.189)	8.0 (0.591)
Dynamic, 7-event, 3-node	1.6 (0.262)	2.2 (0.322)	3.3 (0.353)	1.6 (0.266)	1.9 (0.312)	3.9 (0.519)	1.4 (0.149)	1.7 (0.280)	8.8 (0.592)
Dynamic, 6-event, 4-node	3.4 (0.530)	4.8 (0.416)	5.9 (0.521)	3.1 (0.387)	4.2 (0.362)	8.2 (0.602)	4.8 (0.875)	7.0 (1.528)	79.7 (4.799)
Constrained dynamic, 3-event, 3-node	0.8 (0.224)	1.1 (0.207)	2.7 (0.341)	0.6 (0.127)	1.0 (0.205)	3.0 (0.470)	0.6 (0.188)	0.7 (0.154)	8.2 (0.590)
Constrained dynamic, 5-event, 3-node	0.9 (0.206)	1.3 (0.241)	2.7 (0.228)	0.7 (0.180)	1.1 (0.215)	3.0 (0.240)	0.7 (0.145)	0.9 (0.202)	8.3 (0.510)
Constrained dynamic, 7-event, 3-node	1.6 (0.205)	2.1 (0.302)	3.5 (0.372)	1.4 (0.237)	1.9 (0.360)	4.0 (0.548)	1.5 (0.206)	1.5 (0.204)	9.4 (0.793)
Constrained dynamic, 6-event, 4-node	3.3 (0.491)	4.4 (0.540)	6.1 (0.566)	3.0 (0.434)	3.8 (0.319)	8.5 (0.495)	4.2 (0.684)	5.3 (0.909)	80.6 (6.693)

Table S4. Running times of the graphlet methods for each of the nine network classes in the network classification task. The first row lists three possible node arrival functions. The second row lists three possible edge initiation strategies. The two combined result in $3 \times 3 = 9$ network classes.

Method	AUPR	AUROC	Break-even point	Maximum F-score	Running time, s
Static, 3-node	0.464	0.600	0.456	0.562	9.4
Static, 4-node	0.469	0.610	0.461	0.567	9.4
Static, 5-node	0.464	0.604	0.462	0.566	9.4
Static-temporal, 3-node	0.499	0.644	0.508	0.571	2.7
Static-temporal, 4-node	0.503	0.643	0.609	0.689	2.7
Static-temporal, 5-node	0.482	0.570	0.486	0.554	2.7
Dynamic, 3-event, 3-node	0.479	0.622	0.477	0.569	2.7
Dynamic, 5-event, 3-node	0.474	0.615	0.458	0.569	9.6
Dynamic, 7-event, 3-node	0.470	0.609	0.460	0.572	27.5
Dynamic, 3-event, 4-node	0.541	0.684	0.547	0.594	24.5
Dynamic, 6-event, 4-node	0.525	0.666	0.516	0.583	1,024
Dynamic, 4-event, 5-node	0.591	0.726	0.615	0.620	753
Constrained dynamic, 3-event, 3-node	0.491	0.639	0.498	0.569	**1.1**
Constrained dynamic, 5-event, 3-node	0.492	0.638	0.495	0.570	1.9
Constrained dynamic, 7-event, 3-node	0.492	0.638	0.495	0.571	2.6
Constrained dynamic, 3-event, 4-node	0.550	0.695	0.570	0.600	4.9
Constrained dynamic, 6-event, 4-node	0.550	0.695	0.571	0.600	37.2
Constrained dynamic, 4-event, 5-node	0.594	0.732	0.618	0.637	60.8
Constrained dynamic, 5-event, 6-node	**0.611**	**0.743**	**0.636**	**0.654**	815
Constrained dynamic, 6-event, 7-node	0.608	0.742	0.635	0.652	10,029
Random	0.376 (0.009)	0.495 (0.016)	0.369 (0.007)	0.550 (0.000)	-

Table S5. Detailed node classification results for the different methods and different parameters in each method. The table can be interpreted just as Supplementary Table S3. Notice that in this test of node classification we could test some additional parameters (e.g., graphlets on five or more nodes) compared to the test of network classification (Supplementary Table S3); this is because the test of network classification is computationally much more complex, given that graphlets need to be counted in multiple networks, as opposed to counting graphlets in only one network in the node classification task.

	DyNetAge		BrainExpression2004Age		SequenceAge	
	k_1	k_2	k_1	k_2	k_1	k_2
Static	0.981	0.981	0.947	0.947	0.990	0.990
Static-temporal	0.992	0.992	**0.948**	**0.947**	0.998	0.998
Dynamic	**0.998**	**0.998**	0.926	0.926	**0.999**	**0.999**
Constrained dynamic	0.993	0.993	0.927	0.927	0.991	0.991
Random	0.850	0.851	0.946	0.944	0.914	0.910

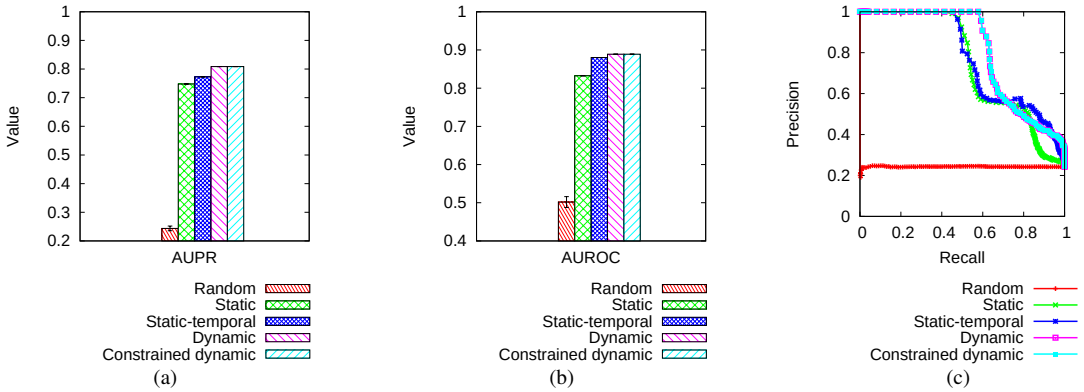
Table S6. Precision of the different methods in the context of aging at the two k values, for the three "ground truth" aging-related data sets (DyNetAge, BrainExpression2004Age, and SequenceAge). For each method, the highest-scoring graphlet size is chosen. In a column, the value in bold is the best result over all methods. Note that even though (constrained) dynamic graphlets are not superior with respect to the two values of k when it comes to BrainExpression2004Age, as illustrated in the table, (constrained) dynamic graphlets are still superior with respect to the entire range of k, as illustrated in Supplementary Fig. S4.

http://www.nd.edu/∼cone/DG/predictions.xlsx

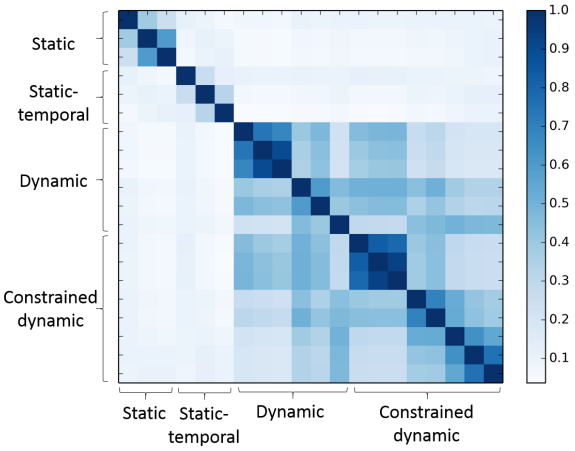
Table S7. Lists of aging-related predictions produced by the four graphlet approaches at the two values of k. Note that we provide a web link to an excel file containing the table with the predictions, as the table is too large to be directly incorporated into the Supplement. Also, note that the parameter versions of the graphlet approaches that were used to make the predictions are not necessarily the same the parameter versions shown in Table 1 in the main paper. Namely, the dynamic graphlet approach at k_1, which had precision of 0.983 in Table 1 in the main paper, resulted in only a single novel aging-related gene, and as such, we had no statistical power to validate this version of the dynamic graphlet approach. Thus, to produce novel aging-related predictions, we used the next highest scoring (in terms of precision) version of the dynamic graphlet approach. Also, we note that one parameter version of static-temporal graphlets produced the maximum precision of 1 with respect to the known aging-related knowledge from DyNetAge for the first value of k. This is interesting, as it means that its predictions are a perfect subset of DyNetAge, despite the two approaches having significant differences: our static-temporal graphlet approach from this study is based on ranking of node pairs and on graphlets only, whereas the DyNetAge study is based on the notion of changing node centralities and on multiple measures of network topology. However, whereas this parameter version of the static-temporal approach resulted in the perfect precision, it has no practical usefulness beyond predicting knowledge that is already predicted by DyNetAge; it cannot predict any novel aging-related genes. For this reason, we left out this version from consideration in our study and instead we have focused on the next highest scoring version of the static-temporal approach, from which we have made novel aging-related predictions.

GO term ID	GO term name
GO:0044281	**Small molecule metabolic process**
GO:0046854	**Phosphatidylinositol phosphorylation**
GO:0046677	**Response to antibiotic**
GO:0050885	**Neuromuscular process controlling balance**
GO:0071356	**Cellular response to tumor necrosis factor**
GO:0006661	**Phosphatidylinositol biosynthetic process**
GO:0001666	**Response to hypoxia**
GO:0007059	**Chromosome segregation**
GO:0048565	**Digestive tract development**
GO:0010043	Response to zinc ion
GO:0071320	Cellular response to cAMP
GO:0006364	rRNA processing
GO:0006805	Xenobiotic metabolic process
GO:0044237	Cellular metabolic process
GO:0044267	Cellular protein metabolic process
GO:0016337	Single organismal cell-cell adhesion
GO:0019886	Antigen processing and presentation ofi exogenous peptide antigen via MHC class II

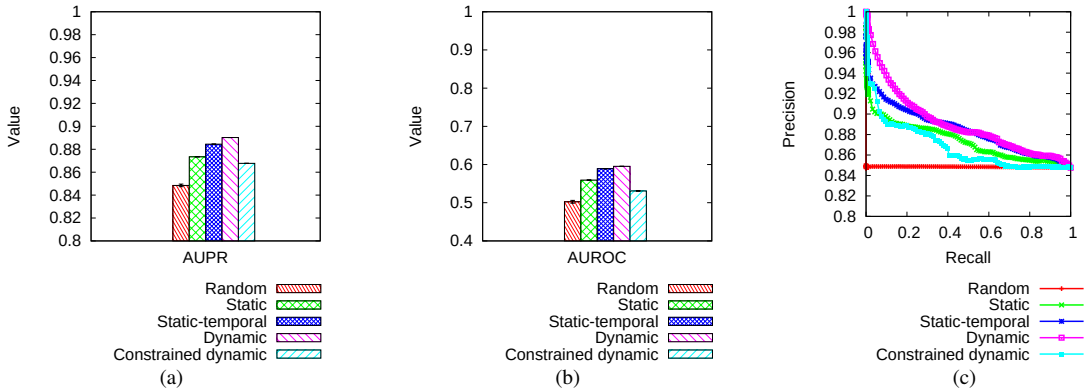
Table S8. GO terms enriched in novel aging-related predictions of dynamic graphlets at the second value of k. GO terms that are also enriched in SequenceAge are shown in bold. All other GO terms are enriched only in dynamic graphets' predictions.



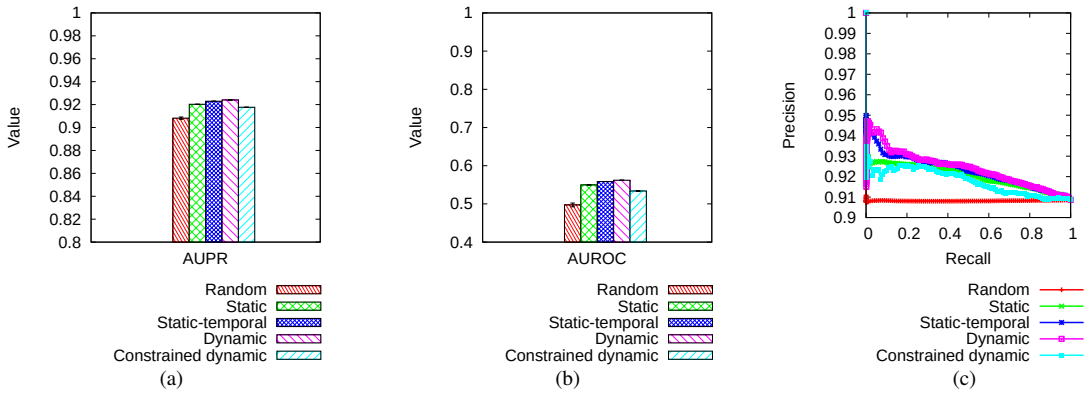
Supplementary Fig. S1. Comparison of the different graphlet approaches in the context of biological network classification, in terms of **(a)** AUPR values, **(b)** AUROC values, and **(c)** precision-recall curves.



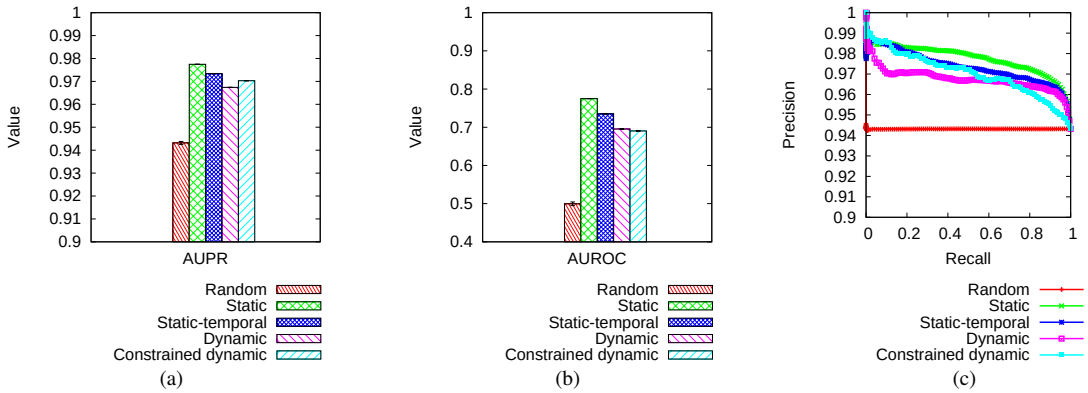
Supplementary Fig. S2. Pairwise similarities between the different methods and their parameter variations in the test of node classification. Similarities are computed as Jaccard coefficients between two methods' top 5% node pairs that are the closest in the graphlet-based PCA space. The order of the methods in the figure directly corresponds to the method order in Supplementary Table S5 (we leave out detailed method names for visual clarity). If we zoom into these results, not only is there a clear separation between static, static-temporal, and (constrained) dynamic graphlets in terms of which nodes they describe as topologically similar, but also, within both dynamic and constrained dynamic graphlets, we can see two clear clusters corresponding to three-node graphlets with different numbers of events. The latter observation suggests that the number of nodes seems to play a larger role in separating the different dynamic graphlet methods compared to the number of events.



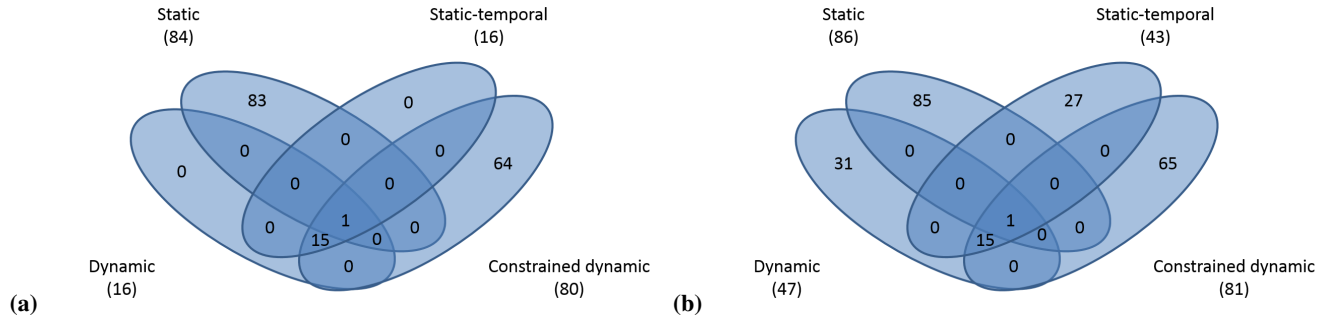
Supplementary Fig. S3. Comparison of the different graphlet approaches in the context of aging-related node classification when considering DyNetAge "ground truth" aging-related data, in terms of **(a)** AUPR values, **(b)** AUROC values, and **(c)** precision-recall curves.



Supplementary Fig. S4. Comparison of the different graphlet approaches in the context of aging-related node classification when considering BrainExpression2004Age, a non-network "ground truth" aging-related data obtained via gene expression data analyses, in terms of **(a)** AUPR values, **(b)** AUROC values, and **(c)** precision-recall curves.



Supplementary Fig. S5. Comparison of the different graphlet approaches in the context of aging-related node classification when considering SequenceAge, a non-network "ground truth" aging-related data obtained via genomic sequence analyses, in terms of **(a)** AUPR values, **(b)** AUROC values, and **(c)** precision-recall curves. Note that even though (constrained) dynamic graphlets are not superior with respect to the entire range of k when it comes to SequenceAge, as illustrated in the figure, (constrained) dynamic graphlets are still superior at the lowest k values (Section 3.4 in the main paper), as illustrated in Supplementary Table S6.



Supplementary Fig. S6. Overlap between aging-related predictions produced by the four graphlet approaches at **(a)** the first value of k and **(b)** the second value of k. The total number of predictions in a given set is shown in parentheses under the set's name.