Bourne *Appl Netw Sci* (2020) 5:88
https://doi.org/10.1007/s41109-020-00329-4

Applied Network Science

RESEARCH Open Access

The spring bounces back: introducing the strain elevation tension spring embedding algorithm for network representation

Jonathan Bourne*

*Correspondence:
jonathan.bourne.15@ucl.ac.uk
University College London, Gower Street, London, UK

Abstract

This paper introduces the strain elevation tension spring embedding (SETSe) algorithm. SETSe is a novel graph embedding method that uses a physical model to project feature-rich networks onto a manifold with semi-Euclidean properties. Due to its method, SETSe avoids the tractability issues faced by traditional force-directed graphs, having an iteration time and memory complexity that is linear to the number of edges in the network. SETSe is unusual as an embedding method as it does not reduce dimensionality or explicitly attempt to place similar nodes close together in the embedded space. Despite this, the algorithm outperforms five common graph embedding algorithms, on graph classification and node classification tasks, in low-dimensional space. The algorithm is also used to embed 100 social networks ranging in size from 700 to over 40,000 nodes and up to 1.5 million edges. The social network embeddings show that SETSe provides a more expressive alternative to the popular assortativity metric and that even on large complex networks, SETSe's classification ability outperforms the naive baseline and the other embedding methods in low-dimensional representation. SETSe is a fast and flexible unsupervised embedding algorithm that integrates node attributes and graph topology to produce interpretable results.

Keywords: Structural network properties and analysis, Community structure in networks, Complex networks in statistical mechanics, Social networks

Introduction

With the rise of social media and e-commerce, graph and complex networks have become a common concept in society. Their ubiquity and the already digitised nature of social networks have led to a great deal of research. Although there is a range of algorithms that can perform supervised learning directly on graphs (Cao et al. 2016; Kipf and Welling 2016; Seo et al. 2018; Scarselli et al. 2009) [see Wu et al. (2020) for a recent survey on the subject], a more common approach is to create embeddings in a latent vector space that traditional supervised learning techniques can then use. These embedding algorithms can either embed the entire graph (Narayanan et al. 2017; Gutiérrez-Gómez and Delvenne 2019) or individual nodes (Grover and Leskovec 2016; Ou et al. 2016; Perozzi et al. 2014; Roweis and Saul 2000); this paper uses only methods that

create embeddings at node level. The embedding algorithms create a vector representations, preserving valuable network properties such as distance on the graph, community structure and node class. These algorithms find embeddings by minimising the distance between similar nodes, within the structure of the network. The literature review by Goyal and Ferrara (2018) provides a survey of the most significant of these algorithms. With recent improvements in neural networks, there has been substantial growth in research on graph embedding algorithms, with over 50 new neural-network-based embedding algorithms published between 2015 and 2020 (Fey and Lenssen 2019). Whilst most methods return Euclidean embeddings, there are also non-Euclidean approaches, such as using hyperbolic space (Nickel and Kiela 2017; Wang et al. 2019). These approaches can outperform Euclidean embeddings when modelling complex and hierarchical structures and can reduce the necessary number of dimensions needed for an effective embedding.

Physics models can also be used to embed graphs in vector space; however, these are typically used only for drawing graphs. While other graph drawing techniques exist (Frick et al. 1995; Koren 2005; Krzywinski et al. 2012), force-directed physics models are some of the most popular (EADES 1984; Fruchterman and Reingold 1991; Kamada and Kawai 1989). These algorithms originated in the 1960s (Tutte 1963), and use simple physics to find an arrangement of nodes that provides an aesthetically pleasing plot of a graph or network. These algorithms are sometimes called 'spring embedders' (Kobourov 2013) due to using springs or spring-like methods to place nodes, and typically attempt to optimise an ideal distance between nodes, minimising the energy of the system. Although popular at the end of the 20th century, spring embedders became less common in research as machine learning became more popular.

The importance of drawing graphs is discussed in several papers (Chen et al. 2018; Matejka and Fitzmaurice 2017; Peel et al. 2018; Revell et al. 2018). These researchers demonstrate that graphs that are structurally very different can appear identical until visualised. A popular statistical equivalent is Anscombe's quartet (Anscombe 1973), a series of four figures showing very different data. However, in terms of the mean, variance, correlation, linear regression and R^2, all four figures in the quartet appear identical. Such visualisations highlight the importance of visualising data and the weaknesses of some commonly used statistical tools.

The Strain Elevation Tension Spring embedding (SETSe) algorithm takes its name from the embeddings it produces. SETSe takes the node attributes of a graph, representing them as a force. The edges are represented by springs whose stiffness is dependent on the edge weight. The algorithm finds the position of each node on a manifold such that the internal forces created by the nodes are balanced by the resistive forces of the springs and the network is in equilibrium. The SETSe algorithm acts as a hybrid between the advanced techniques of the machine learning graph embedders used for analysis and the intuitive simplicity of the spring embedders used for graph drawing. SETSe functions in a different way to most graph embedding methods. Whilst most algorithms allow the user to choose the dimensionality of the embedded results, SETSe returns a manifold of a fixed number of dimensions that is the same as the number of node features plus the graph space. This effectively means that whilst most graph embedding algorithms use a latent embedded space, SETSe uses an explicit feature space using the node features to

extend the number of dimensions the graph occupies; in some ways, this is similar to a kernel method. The final key difference is that SETSe does not try and place similar nodes close together; as such, there is no loss function in the conventional sense. The original purpose of SETSe was to provide a metric of robustness for power networks to cascading failure. The algorithm can also be applied to understanding conflict in social networks, and predicting culture in organisations amongst others. This paper is therefore used to introduce the method and basic functioning of SETSe.

This paper demonstrates that in a world of sophisticated machine learning, there is still a role for simple, intuitive embedding methods. It shows that SETSe can find meaningful embeddings for the Peel's quintet (Peel et al. 2018) series of graphs, as well as the relations in Facebook data (Traud et al. 2012). It finds these embeddings efficiently in linear iteration time and space complexity. An R package has been created that provides all the functionality necessary to run SETSe analysis/embeddings (available from https://github.com/JonnoB/rSETSe).

The paper asks can SETSe distinguish between graphs that are identical using traditional network metrics? It also asks can SETSe be used to classify individual nodes? To gauge how well SETSe performs these tasks, it is compared against several popular graph embedding algorithms: node2vec (Grover and Leskovec 2016), SDNE (Wang et al. 2016), LLE (Roweis and Saul 2000), Laplacian Eigenmaps (Belkin and Niyogi 2003), HOPE (Ou et al. 2016) and one deep graph convolutional embedding method DGI (Velič ković et al. 2018).

Method

This section begins by providing a simple example of the SETSe algorithm. It then describes and defines the physics model that underpins the embedding method. The algorithmic implementation and practical issues related to convergence are also briefly described. The datasets used in this paper are then introduced. Finally, the analyses performed are described.

Introduction to SETSe: a simple example

The SETSe algorithm takes a network $\mathcal{G}$ containing the set of V nodes and the set E edges. It converts the n attributes or variables of each node into orthogonal forces where each attribute occupies a single dimension. In each dimension, the total sum (across the network) of the forces in that dimension is 0 (i.e. the network is balanced in all dimensions). Each edge is converted to a spring whose stiffness k is taken from an attribute of the edge, typically the edge weight. If no edge attribute is to be used, all the edges in the network take k as an arbitrary constant. The algorithm then positions each node on an $n+1$-dimensional manifold such that no node experiences a net force. Like other spring embedders (Fruchterman and Reingold 1991; Kamada and Kawai 1989; Quigley and Eades 2001), SETSe is subject to the n-body problem (Aarseth 2003; Springel et al. 2005) and must be solved iteratively.

The functioning of SETSe is best described using example. Consider the simple network shown in Fig. 1. The network is planar, and so can be drawn in two dimensions (x and y), with no edges crossing. The nodes in the network have a single attribute/variable that acts perpendicular to the plane; as such, the nodes in the network act as beads

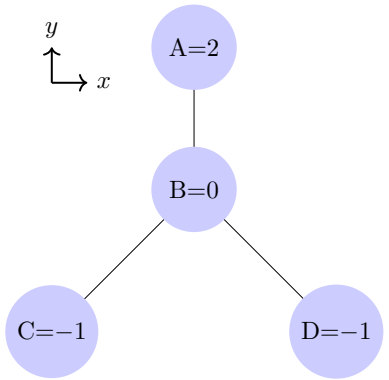

Fig. 1 A network of four nodes and three edges. The node attributes are considered forces that act in the *z*-direction. The forces acting on the network balance, but the forces acting on the nodes only balance when the nodes are in the appropriate position in the *z*-axis; the network is shown in the *x*–*y* planes

whose movement is restricted to the z-axis and are fixed in x and y. The nodes in the network have an identical mass m, and the edges between the nodes have a common distance d, which is the length of the spring at rest. Node A exerts a force of 1, node B exerts no force on the network, while nodes C and D exert a force of -1 each, resulting in a net force of 0. The edges of the network are springs that, when stretched, act according to Hooke's law $F = \Delta Hk$, where ΔH is the extension of the edge and k the spring stiffness such that $0 < k \leq \infty$.

Although the vertical forces (defined by the node attribute) across the network sum to 0, the individual nodes are not in equilibrium and so begin to move in the direction of their respective forces. The vertical distance between node pairs resulting from this movement extends the springs, creating a resistive force. The network will find a three-dimensional equilibrium when the net force acting on each node is 0. This occurs when the elevation of the nodes in the system is such that for each node, the sum of the vertical tension in all edges connected to that node is equal and opposite to the force produced by the node itself. As an example, if $k = 1000$ and $d = 1$, the equilibrium positions of the nodes are 0.1450, 0.0185, -0.0818 and -0.0818 for the nodes A to D, respectively. The interested reader can confirm that the net forces on each node are 0, using Pythagoras' theorem and Hooke's law.

A crucial point in this example is that the nodes are beads. As such, the xy positions are fixed and only the vertical component of the spring tension has an impact on the nodes; all horizontal forces can be disregarded. Ignoring the horizontal force means that the xy position of the nodes can be ignored, reducing the initial layout of the network to a zero-dimensional point in space, out of which a one-dimensional node elevation appears. The horizontal distance between nodes is reduced to a crucial but abstract mapping value.

SETSe space is a non-Euclidean metric space of $n + 1$ dimensions where n is the number of attributes the network has. The $n + 1$th dimension is the graph space, representing the graph adjacency matrix. The distance between connected nodes in the graph space is $d_{i,j}$. Dimensions 1 to n are Euclidean, while the graph dimension is not. This concept is visualised in Fig. 2, which shows nodes embedded in pairwise two-dimensional space

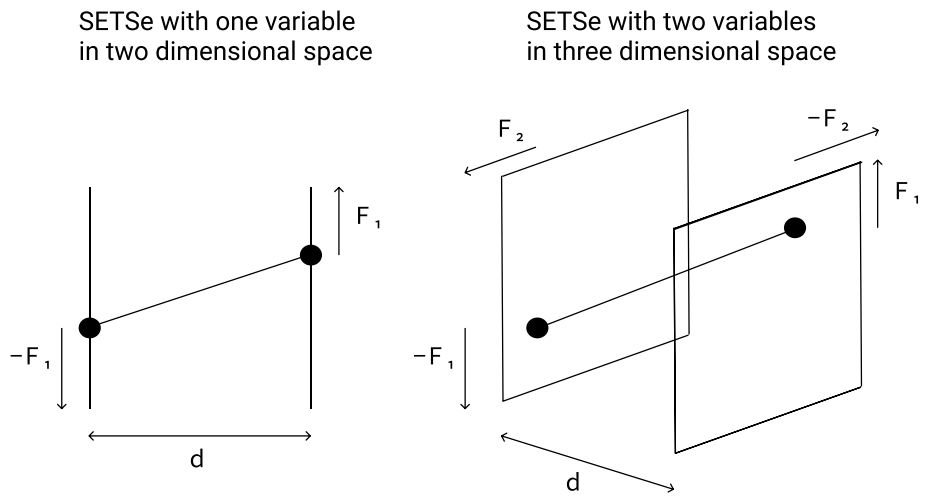

Fig. 2 Two nodes in a network in two-dimensional and three-dimensional SETSe systems. The minimum distance *d* between the nodes is maintained by the graph space dimension

and pairwise three-dimensional space. The graph space acts as the minimum distance between the nodes. Figure 2 shows that the nodes occupy parallel Euclidean hyperplanes separated by the graph space. As SETSe space is locally Euclidean, it is an $n+1$ -dimensional manifold. As an example of a network that is pairwise Euclidean but not Euclidean overall, consider a maximally connected network of four nodes. There is no arrangement of the nodes on a plane where the distance between all nodes is equal, although pairwise all nodes can have the same distance. As the example implies, SETSe can produce entirely euclidean embeddings if the network adjacency matrix can be represented in Euclidean space, such networks include planar networks. However, such networks are special cases, and the generalised non-Euclidean adjacency matrices only are discussed here.

Creating the physics model

The calculation of the solution of a single variable graph is described below. The extension for higher dimensions is described in the subsequent paragraph. The net force acting on node i can be written as $F_{\text{net},i} = F_i - F_{\text{vten,i}}$, where F_i is the force produced by the node and $F_{\text{vten,i}}$ is the vertical component of the net tension acting on the node from the springs. The net force on node i is shown again in Eq. 1 where $F_{\text{ten,i,j}}$ is the total tension in edge i, j. The angle $\theta_{i,j}$ is the angle of the force between nodes i and j. The tension in an edge is given by Hooke's law as $F_{\text{ten,i,j}} = k_{i,j}(H_{i,j} - d_{i,j})$, where $H_{i,j}$ is the length of the extended spring and $d_{i,j}$ is the graph distance between the nodes. The length of the extended spring length $H_{i,j}$ can be found, as it is the hypotenuse of the distance triangle between nodes i and j such that $H_{i,j} = \sqrt{\Delta z_{i,j}^2 + d_{i,j}^2}$, where $\Delta z_{i,j}$ is the elevation difference between nodes i and j, $\Delta z_{i,j} = z_j - z_i$. As $\cos\theta = \frac{\Delta z}{H}$, the equation for net tension can be rearranged into an alternative expression of edge tension, which is shown in Eq. 3. The strain component of SETSe is simple mechanical strain and is shown in Eq. 4. Strain and tension are perfectly correlated in the special case that k is constant for all edges in the network.

$$F_{\text{net},i} = F_i - \sum_j^n F_{\text{ten},i,j} \cos\,\theta_{i,j} \tag{1}$$

$$F_{\text{net},i} = F_i - \sum_j^n k_{i,j}(H_{i,j} - d_{i,j})\frac{\Delta z_{i,j}}{H_{i,j}} \tag{2}$$

$$F_{\text{net},i} = F_i - \sum_j^n k_{i,j}\Delta z_{i,j}\left(1 - \frac{d_{i,j}}{H_{i,j}}\right) \tag{3}$$

$$\varepsilon_{i,j} = \frac{H_{i,j} - d_{i,j}}{d_{i,j}} \tag{4}$$

The extension of SETSe from a graph with a single attribute to a graph with n attributes is straightforward. The hypotenuse vector is $\mathbf{H}_{i,j} = \mathbf{z}_i - \mathbf{z}_j + \mathbf{d}$, which is a vector of $n+1$ elements, where the first n elements are the differences in position between nodes i and j in the n dimensions, and the $n+1$th dimension is the graph distance d. The scalar length of $\mathbf{H}_{i,j}$ is the Euclidean distance between the nodes in $n+1$-dimensional space, i.e. $H_{i,j} = \sqrt{(\sum_1^n (z_{i,q} - z_{j,q})^2 + d_{i,j}^2)}$. To find the angle between the hypotenuse and the distance between the two nodes in dimension q, the cosine similarity is used, as shown in Eq. 5. As all entries of $\Delta\mathbf{z}_{q,i,j}$ are 0, apart from the qth entry, the cosine similarity simplifies to Eq. 6, which is the 'vertical' distance between the nodes in dimension q over the scale length of $H_{i,j}$. It is then easy to see that Eq. 7 is the multidimensional equivalent of Eq. 3.

$$\cos\,\theta_{q,i,j} = \frac{\mathbf{z}_{q,i,j} \cdot \mathbf{H}_{i,j}}{\|\mathbf{z}_{q,i,j}\|\|\mathbf{H}_{i,j}\|} \tag{5}$$

$$\cos\,\theta_{q,i,j} = \frac{\Delta z_{q,i,j}}{H_{i,j}} \tag{6}$$

$$F_{\text{net},q,i} = F_{q,i} - \sum_j^n k_{i,j}\Delta z_{q,i,j}\left(1 - \frac{d_{i,j}}{H_{i,j}}\right) \tag{7}$$

The distance d between the nodes is a key parameter when it comes to finding the final elevation embedding. If the distance is not a constant, it must be meaningful for the type of network being analysed. As an example, the distance in metres between two connected points on an electrical circuit is unlikely to be meaningful; however, a variable distance may be appropriate if traffic were being analysed. Understanding meaningful distance metrics is not explored in this paper, and distance between nodes is considered a constant across all edges. It should also be noted that although SETSe is embedded in Euclidean space and the graph space, other non-Euclidean spaces can be used. For example, the hyperbolic space used by Nickel and Kiela (2017) could be used instead of

Euclidean space as Newtonian physics is still valid; whilst such options are intriguing, they are beyond the scope of this paper.

SETSe can be used on continuous and categorical variables. In both cases, the forces must be balanced; this is when the net value of the sum of the forces across all nodes equals 0. For continuous variables, the raw attribute force of node i (F_i) is normalised to create balanced force $F_{i,\text{bal}}$ by subtracting the mean from all values, as shown in Eq. 8, where $|\mathcal{V}|$ is the number of nodes in the network.

$$F_{i,\text{bal}} = F_i - \frac{1}{|\mathcal{V}|}\sum_{1}^{|\mathcal{V}|} F_i \tag{8}$$

For categorical node attributes, each level is treated as a binary attribute, making as many new dimensions as there are levels; this is similar to how variables in linear and logistic regression are treated. The total force level dimension γ is the fraction that level makes up of the total number of nodes, as shown in Eq. 9. The force produced by the nodes in each level dimension can then be treated as continuous variables, as described previously in Eq. 8.

$$F_\gamma = \frac{|\mathcal{V}_\gamma|}{|\mathcal{V}|} \tag{9}$$

With the force and distance relationship between pairs of nodes defined, it is now possible to look at the method used to find the equilibrium state of the network and its corresponding strain, elevation and tension embeddings. The difficulty in solving such a problem is that the relationship between the final elevation of a node and the force it experiences is non-linear. In addition, each node is affected by all other nodes and spring stiffness k in the network. This interaction creates a situation that is similar to the n-body problem of astrophysics. In this case, although the nodes act as bodies, instead of exerting a force on all other nodes, as celestial bodies do, they exert a force only on those nodes with which they have a direct connection.

The equilibrium solution can be found by treating the problem as a dynamic system and iterating through discrete time steps until the system reaches the equilibrium point. By representing the network as a dynamic system, the acceleration and velocity of each node must be calculated. Using Newton's second law of dynamics, $F = ma$, where F is the force acting on the node and a is the acceleration, the nodes need to be assigned an arbitrary constant mass m (note mass does not affect the final embeddings). The net force acting on each node at time step t is then Eq. 10, where F is the force generated by the node according to the node attribute. The system is assumed to be a viscous laminar fluid, and so the damping is simply the product of the velocity v and the coefficient of drag c. Friction is used to cause the system to slowly lose energy and converge. While it does not affect the value at convergence, it needs to be correctly parametrised or the system will not converge. This is discussed in the Additional file 1: Appendix.

$$F_{\text{net,i}} = F_{\text{vten}_\text{i}} + F_i - cv_i \tag{10}$$

Knowing the net force acting on the node allows calculation of the equations of motion at each time step. Velocity can be calculated as $v_t = v_{t-1} + \frac{F_{\text{net},t}}{m}\Delta t$, where

v is the velocity at time t. Distance is the elevation embedding and is calculated by $z = v_{t-1}\Delta t + \frac{1}{2}a_t\Delta t^2 + z_{t-1}$.

The SETSe algorithm is shown in algorithm 1. The equations described in Eqs. 1–10 are either converted to vectors or matrices, allowing all nodes and edges in the network to be updated simultaneously. The algorithm takes a graph $\mathcal{G}$, which has been processed so that each edge has distance $d_{i,j}$ and spring constant $k_{i,j}$. The dynamics of the network are all initialised at 0; only the forces exerted by the nodes are non-zero values. In algorithm 1, vectors are lower-case letters in bold, while matrices are in bold and capitals. The time in the system is represented by t and the time step per iteration is Δt. The elevation of each node in the system is represented by the matrix $\mathbf{Z}$. The elevation difference across each edge is $\Delta\mathbf{Z}$, and is obtained by subtracting the transpose of the elevation matrix from the original elevation matrix. The hypotenuse, or total length, of the edge is represented by the matrix $\mathbf{H}$ and is found using the elevation difference $\Delta\mathbf{Z}$ as well as the horizontal difference $\mathbf{d}$. The vertical component of the tension in each edge is represented by $\mathbf{F}_{vten}$ and is the element-wise product of the edge spring stiffness matrix $\mathbf{K}$ with the extension of the edge; this matrix is then multiplied element-wise again using the element-wise product by the tangent of the angle of the edge. Line 8 shows that the vertical component of the force $\mathbf{f}_{vten}$ is updated by summing the rows of each line in the $\mathbf{F}_{vten}$ matrix using a column vector of 1s, that is, $|\mathcal{V}|$ long. The elevation of each node is updated on line 9 of the algorithm. The vector $\mathbf{z}$ is then reshaped using a function into matrix form. Line 11 updates the velocity of each node in the network. Line 12 updates the static force on each node $\mathbf{f}_{static}$. Static force is the force exerted by the node minus the sum of the tensions exerted by all the connected edges. The next update is the system friction or drag $\mathbf{f}_d$. The system force $\mathbf{f}_{net}$ is then updated. Finally, the acceleration $\mathbf{a}$ to be used in the next iteration is calculated.

One of the advantages that SETSe has over traditional force expansion algorithms (Kamada and Kawai 1989; Fruchterman and Reingold 1991) is that the distance from the optimal solutions is known. In the other algorithms, the loss function is to reduce the total energy of the system to some unknown minimum. However, SETSe has a loss function more similar to the error metrics used in statistics or machine learning. The ideal static force of the system is 0 and the initial static force is $\sum \|F_i\|$, which is thus bounded in a finite space. This is an important consideration when it comes to efficient convergence and auto-convergence and is discussed further in the Additional file 1: Appendix. Although the stop condition of the algorithm is that the static force in the network is 0, in practice, the system is said to have converged if $\mathbf{f}_{static} \approx 0$. In this paper, the tolerance for convergence will be $f_{static} \leq \frac{\sum \|F_i\|}{10^3}$, that is when the static force is reduced to 1/1000th of the absolute sum of forces exerted by the nodes.

Algorithm 1: The SETSe algorithm

Result: The strain, elevation and tension embeddings of the original graph

$t = 0$;
$\mathbf{f}_{static} = \mathbf{f}$;
while $\sum \|\boldsymbol{f}_{static}\| \neq 0$ **do**
  $t = t + \Delta t$;
  $\Delta\mathbf{Z} = \mathbf{Z} - \mathbf{Z}^T$;
  $\mathbf{H} = \sqrt{\Delta\mathbf{Z}^2 + \mathbf{d}^2}$;
  $\mathbf{F}_{vten} = \mathbf{K} \circ (\mathbf{H} - \mathbf{d}) \circ \frac{\Delta\mathbf{Z}}{\mathbf{H}}$;
  $\mathbf{f}_{vten} = \mathbf{F}_{vten}\mathbf{J}_{|\mathcal{V}|}$;
  $\mathbf{z} = \mathbf{v}\Delta t + \frac{1}{2}\mathbf{a}\Delta t^2 + \mathbf{z}$;
  $\mathbf{Z} = \mathbf{f(z)}$;
  $\mathbf{v} = \mathbf{v} + \mathbf{a}\Delta t$;
  $\mathbf{f}_{static} = \mathbf{f} - \mathbf{f}_{vten}$;
  $\mathbf{f}_d = c_d\mathbf{v}$;
  $\mathbf{f}_{net} = \mathbf{f}_{static} - \mathbf{f}_d$;
  $\mathbf{a} = \frac{\mathbf{f}_{net}}{m}$;
end

Practical convergence issues

The implementation of the algorithm in the R package `rSETSe` has two modes: sparse and semi-sparse. Semi-sparse mode is used on smaller graphs, and sparse mode is for larger graphs (starting at 5000–10,000 edges); the complexity of sparse mode is linear to the number of edges $\mathcal{O}(|\mathrm{E}|)$. The mode has no impact on the final embeddings. Time and space complexity are discussed further in "Complexity" section.

Although there are several parameters of the physical model that must be initialised before running the algorithm, only two have any real bearing on the final embeddings. Drag, time step and mass affect the rate of convergence, but not the final outcome (when $F_{\text{static}} = 0$). The distance between the nodes affects the outcome as the final elevation will change. However, when $F_{\text{static}} = 0$, the angle between the nodes is unaffected by the length of the edges and so elevation can be normalised. Only the force variable and the spring stiffness have an impact on the final converged values. The force variable is not controlled by the user, leaving only k. As such, the value of k must be constant for all networks under evaluation, or if k is a function, then $k = \mathrm{f}(x)$ must be consistently parametrised.

All networks in this paper are embedded using bi-connected SETSe, a more advanced method than algorithm 1. This method breaks the network into bi-connected sub-graphs then calls auto-SETSe, which is an algorithm that chooses the coefficient of drag using a binary search. Bi-connected SETSe and auto-SETSe are discussed in detail in the Additional file 1: Appendix.

Data

Two datasets are used in this paper to illustrate how SETSe works and how it can be used to gain insight into network structure and behaviour. Both datasets have binary edges, meaning that k is constant across all edges in all networks. This reduces the embeddings from three to two, as strain and tension have a perfect linear relationship.

Peel's quintet

Peel's quintet (Peel et al. 2018) is an example of the graph equivalent of Anscombe's quartet (Anscombe 1973). It is a collection of five binary attribute graphs that have an identical number of nodes, edges connections between and within classes, and assortativity. The networks are very different when visualised (see Fig. 3). In (Peel et al. 2018) the authors achieve this situation by dividing the binary classes into two sub-classes, which have different mixing patterns. They then develop an alternative metric and demonstrate that it can distinguish between the quintet and other network structures. Peel's quintet is essentially a hierarchical stochastic block model. Each network has two blocks containing two sub-classes. Each sub-class contains 10 nodes, with a total of 40 nodes per network. Each network has 160 edges with 80 edges connecting the classes together and 80 edges internally in each class. Because the number of edges connecting within and between the sub-class is distinct, the overall network structure is itself distinct, even though in terms of traditional network metrics they are identical. Peel's quintet will be used as an example of how SETSe is affected by graph topology and the network attributes, in this case, the two known communities and the hidden communities.

Figure 3 shows Peel's quintet. The classes are shown as being either turquoise or red, while the two hidden classes are triangles or circles. While type A is simply a random network, the other networks show varying types of structure produced by the inter-hidden group connection patterns. Table 1 shows the block models the network is based on.

Facebook data

The Facebook 100 dataset by Traud et al. (2012) is a snapshot of the entire Facebook network on a single day in September 2005. At this time, Facebook was open only to 100 US universities. There were very few links between universities then, so each one can be considered a stand-alone unit. Such an assumption would not be possible now. The data allow

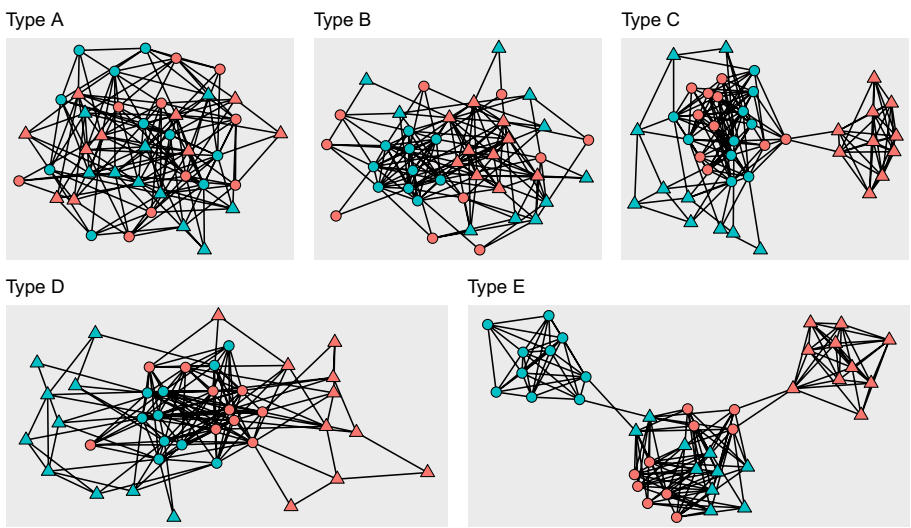

Fig. 3 These networks were introduced by Peel et al. (2018). They are all networks that have identical numbers of nodes edges, group size, within-group connections and between-class connections. However, the networks are clearly structurally distinct

Table 1 Block model for Peel's quintet

(a) Type A

	a1	a2	b1	b2
a1	10	20	20	20
a2	-	10	20	20
b1	-	-	10	20
b2	-	-	-	10

(b) Type B

	a1	a2	b1	b2
a1	38	2	20	20
a2	-	0	20	20
b1	-	-	0	2
b2	-	-	-	38

(c) Type C

	a1	a2	b1	b2
a1	38	2	0	0
a2	-	0	80	0
b1	-	-	10	20
b2	-	-	-	10

(d) Type D

	a1	a2	b1	b2
a1	10	20	0	0
a2	-	10	80	0
b1	-	-	10	20
b2	-	-	-	10

(e) Type E

	a1	a2	b1	b2
a1	38	2	0	0
a2	-	0	80	0
b1	-	-	0	2
b2	-	-	-	38

insight into the structure of relationships in attributed social networks. The networks are anonymised and the universities are referred to by a reference. Caltech36 is the smallest university network and has only 769 nodes and 16,656 edges, while Penn94 has the most nodes with 41,554 and Texas84 has the most edges with 1,590,655. The original study on this dataset (Traud et al. 2012) found there were assortativity patterns within the variables that were generally common across all universities, such as tendency to be connected to students who will graduate at the same time or who lived in the same university accommodation. The networks have seven attributes, all of which are categorical and have been anonymised. These attributes are: student type, gender, major, minor, dorm, year of graduation and high school.

Experimental analysis

The experiments are broken across the two datasets. The first set of experiments will focus on Peel's quintet, distinguishing network types, then distinguishing between node types. The second set of experiments will look at the Facebook data, first comparing assortativity with SETSe, which is similar to distinguishing between networks, then distinguishing between node types on the Facebook dataset. The two SETSe dimensions will be elevation and node tension. Node tension is the mean absolute tension in the edges connected to the nodes $\nu_{\text{ten,i}} = \frac{\sum F_{\text{ten},i,j}}{n}$, where for this expression only, n is the number of edges for node v_i.

This paper uses four accuracy metrics: accuracy (Eq. 11), balanced accuracy (Eq. 12), f1 score (Eq. 13) and Cohen's kappa (Eq. 14), where P, N, TP, TN, FP and FN are the number of positives, negatives, true positives, true negatives, false positives and false negatives, respectively. TPR is the false positive rate, $\text{TPR} = \frac{\text{TP}}{P}$, and TNR is the true negative rate, $\text{TNR} = \frac{\text{TN}}{N}$. p_o is the observed probability of two events occurring, e.g. predict class one truth is class one. p_e is the expected probability of two events occurring, given their overall prevalence.

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}} \tag{11}$$

$$\text{BAL\_ACC} = \frac{\text{TPR} + \text{TNR}}{2} \tag{12}$$

$$\mathrm{f1} = \frac{2\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \tag{13}$$

$$\kappa = \frac{p_o - p_e}{1 - p_e} \tag{14}$$

Distinguishing between networks

The first analysis of the performance of SETSe will be to compare it to a selection of other node embedding methods using Peel's quintet (Peel et al. 2018). The methods that it will be compared against are node2vec (Grover and Leskovec 2016), SDNE (Wang et al. 2016), LLE (Roweis and Saul 2000), Laplacian Eigenmaps (Belkin and Niyogi 2003), HOPE (Ou et al. 2016) and DGI (Velič ković et al. 2018). The methods cover three main areas: graph factorisation (Roweis and Saul 2000; Belkin and Niyogi 2003; Ou et al. 2016), random walks (Grover and Leskovec 2016) and deep learning (Wang et al. 2016; Velič ković et al. 2018). It should be noted that of the seven alternative methods, only DGI uses node features as part of the embedding. A set of 100 networks from each of the five classes of Peel's quintet will be generated and embedded into two dimensions using each of the embedding methods. The embeddings are produced at node level and will be aggregated using the mean to network level. The linear separability of the aggregated embeddings for each class will be compared.

Classifying class and sub-class of Peel's quintet

This section will test the ability of the embedding methods to separate the hidden classes of Peel's quintet. The embeddings generated in the previous experiment will be used, and a multinomial logistic regression will be created for each network to see the accuracy of separating either the nodes into their known classes or into their hidden classes. In theory, SETSe should be able to almost trivially separate the classes as the algorithm is class aware; however, separating the sub-classes is less clear. The logistic regression will use as the independent variables the two embedding values at node level that were generated when distinguishing between network types. In the case of SETSe, these will be the node elevation and the mean node tension. The role of the logistic regression is not so much to be a predictive model but to test the separability of the data; as such, no cross validation will be necessary. The accuracy measure will be accuracy as the classes are balanced. The performance of SETSe will be compared against all other embedding methods. As each node's relation to the rest of the network is being analysed using elevation and mean edge tension the graph space is effectively removed; as a result, the data are no longer non-Euclidean and logistic regression can be used.

Relationship with assortativity

The Facebook data will be embedded for all of the 100 universities using the graduation year of the student. Although technically categorical data, graduation year can be treated as continuous and will be done so for speed of calculation. Missing data will not exert a force. The embeddings will be aggregated to network level using the mean elevation and mean node tension. The resulting two-dimensional data will be compared to the

assortativity scores of the data to see if there is a relationship between the SETSe embeddings and the network assortativity.

Predicting node class in Facebook data

This tests to see how well SETSe can separate the classes within large and complex networks. Year of graduation will be the embedding class, student type will be the hidden sub-class. The two main classes of student are types 1 and 2, which make up 80% and 15% of the dataset, respectively. The meaning of student type is not clear from the original paper, but it appears to be graduate students or alumni. Due to the distribution of student type 2, only 2005 will be used for the hidden sub-class test. Student type is being chosen over the other variables, as dorm, major, minor and high school have so many levels that embedding would be impractical. Gender has two levels only; however, there is almost no assortativity suggesting a complete lack of structure.

Due to the complexity of the data, a k-nearest-neighbour approach will be used. This method will label the node as the majority class of the nearest k nodes in SETSe space. The nearest-neighbour model will be compared with graph adjacency voting. Graph adjacency voting finds the majority class amongst all nodes for which the target node shares an edge. The graph adjacency voting will use the full network, but only student types 1 or 2 will count towards the totals. The metrics used to evaluate performance will be accuracy, balanced accuracy, Cohen's kappa and the f1 score. The hidden classes are highly imbalanced in some of the universities, and so the results need to be interpreted with care. The hidden class model accuracy will also be compared against the naive ratio of type 2 students (the majority class) to all students.

Computational details

Each simulation used a single core Intel Xeon Gold 2.3 GHz processor with 16 GB of RAM, and maximum wall clock time per simulation was limited to 12 hours. Code and analysis used R version 4.0, and made extensive use of `igraph` (Csardi and Nepusz 2006) and `rSETSe` https://github.com/JonnoB/rSETSe packages. HOPE, LLE, Laplacian Eigenmaps and SDNE embeddings were performed using the `GEM` library (Goyal and Ferrara 2018), node2vec was performed using the pip installable implementation from https://github.com/eliorc/node2vec and DGI used the Stellargraph library (Data61 2018). The Python version was 3.8.

Results

Peel's quintet

The first test of the SETSe algorithm uses Peel's quintet of networks. The 100 networks of each class are projected into SETSe space then aggregated using the mean absolute elevation and the mean of the node tension. Figure 4 shows the results of the six algorithms reducing the networks to two-dimensional space. It is clear that the different connection patterns between the nodes result in distinctive tension elevation patterns within the graph class. This results in the five networks being trivially separable in SETSe space. The other graph embedding algorithms struggle to differentiate the network types, node2vec is the most successful projecting the graph types into more or less concentric quarter rings in two dimensions. The HOPE algorithm also has some success,

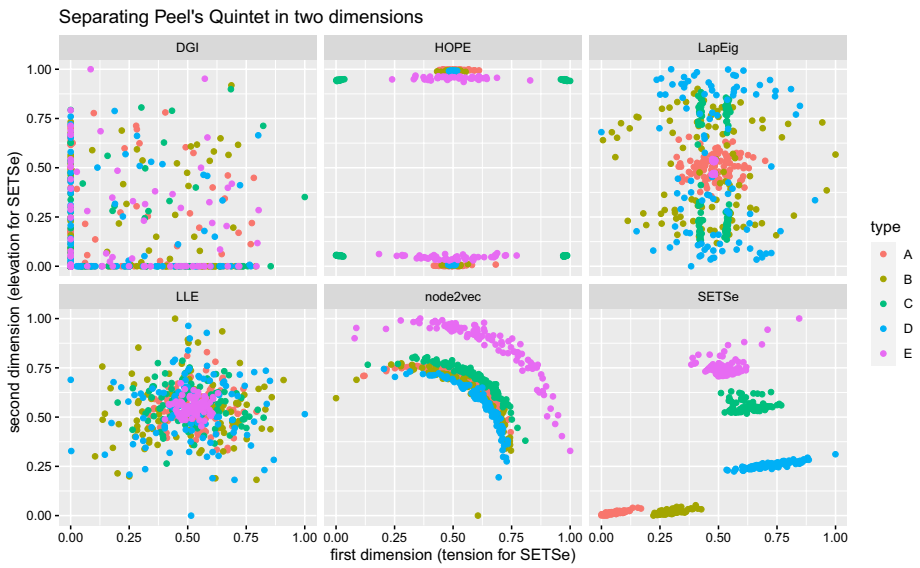

Fig. 4 One hundred examples of each of the five network classes of Peel's quintet introduced by Peel et al. (2018). The networks are clearly linearly separable in the averaged SETSe space even though all 500 are identical using assortativity

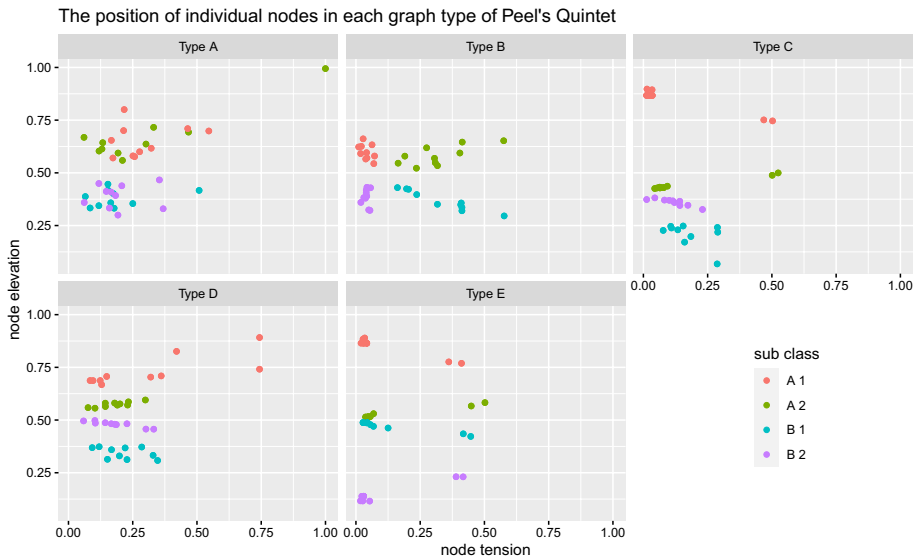

Fig. 5 Embeddings of individual nodes; the *x*-axis is node tension and the *y*-axis is node elevation. This representation reveals the hidden structure of the groups in all network types apart from type A

but like node2vec failed to provide a clear linear separability. The SDNE algorithm was unable to provide successful embeddings; this may be due to the number of embedding dimensions being so low, or due to the structure of the quintet graphs themselves.

Clearly, SETSe is successful at separating the graph types. However, it is also interesting to know whether it can assign the nodes to the correct sub-classes within each graph type. Figure 5 shows the SETSe embedding of the nodes in the elevation tension dimensions for an example graph of each type. The nodes are coloured by the hidden sub-class.

As can be seen, there are clear patterns in the node placement. Although the sub-classes of type A cannot be distinguished, types C, D and E appear to be linearly separable in the elevation dimension alone. In contrast, type B produces a roughly symmetrical distribution requiring elevation and strain for separation. The separability is checked for all 500 networks using SETSe and the five other embedding types. The results are shown in Fig. 6. The figure shows how each embedding technique separates the classes and sub-classes for each type of graph in Peel's quintet. A multinomial logistic regression with two independent variables, reflecting the two dimensions of the embedding, was used to model the accuracy of each class and sub-class within the graphs. The SDNE algorithm was not included in the figures due to poor performance at low dimensions.

As can be seen from Fig. 6, SETSe outperforms the other embedding algorithms in every case, again node2vec and HOPE come next in terms of performance. Despite being class aware, DGI could not separate the classes or sub-classes very well at such low dimensions. When comparing pure linear separability, SETSe greatly outperforms all other embedding techniques. With the exception of identifying the sub-class of graph type A, SETSe can linearly separate the classes and sub-classes at least 67% of the time. And it can perfectly linearly separate the sub-classes in four of the ten cases. No other embedding method is comparable to SETSe; DGI can linearly separate all four sub-classes for graph E 39% of the time, whilst HOPE separates D 32% of the time, but for most cases, linear separation is not possible on either the known binary class or the four hidden sub-classes. In addition, 25% of the graphs failed to converge with the DGI embedding. When the other methods are allowed to embed the 20 node graphs in eight dimensions, their performance increases substantially to a level comparable with SETSe. The failure of SETSe to have perfect class separation in types B, C and E comes from specific nodes having so many neighbours of the opposite class that they are pulled below the class line. As an additional comparison, three community detection algorithms were

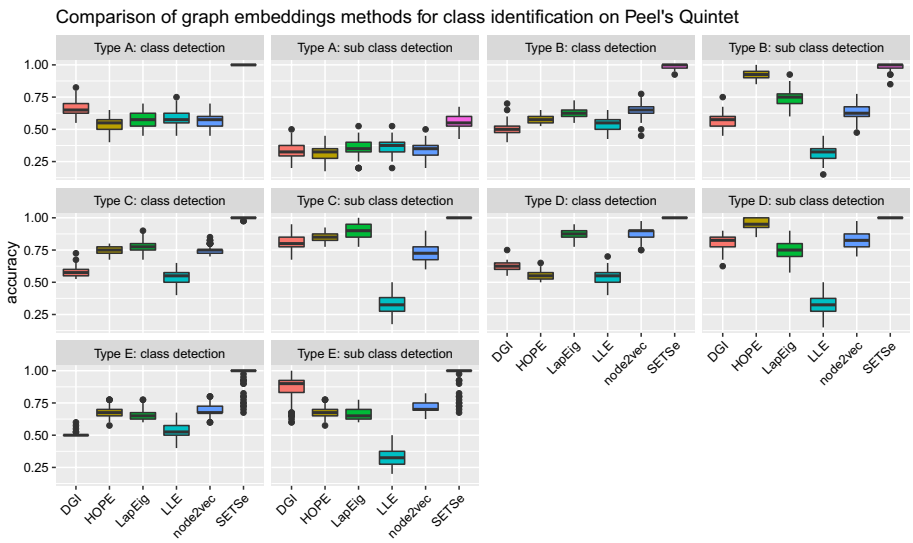

Fig. 6 Using the knowledge of the groups, SETSe greatly outperforms the other node embedding techniques, at separating the classes and sub-classes for each type of Peel's quintet. The upper and lower bounds on the boxes describe the 26th and 75th percentiles. The whiskers are 1.5 times the inter-quartile range

also compared: Fast Greedy (Clauset et al. 2004) , Walktrap (Pons and Latapy 2006) and Louvain (Blondel et al. 2008). These algorithms were not successful at distinguishing between the classes or sub-classes.

Analysing Facebook data using SETSe

The Facebook 100 dataset was embedded into SETSe space using the variable graduation year. The results for four different universities are shown in Fig. 7. For ease of viewing, the x-axis is presented on a log scale. Figure 7 shows three universities with low assortativity (Auburn, Maine and Michigan), meaning that there is a high degree of mixing between years, and one university with a high assortativity (BC), meaning students tend to associate within years. Although it is not possible to explain why the universities have these differences [Peel et al. (2018) suggest it is due to housing allocation policy], it is possible to analyse the relation between assortativity and SETSe. It is clear from the figure that the years separate to roughly their own tension elevation band within the data. This is what we would hope to see given that the embedding uses force based on the graduation year. The overall shape of the data is a funnel with the 'nose' at the low-tension end and the funnel at the higher-tension end. It is clear from the plot that the younger years (those graduating in 2008 and 2009) are more separate than the older years (graduation in 2004, 2005 and 2006). This is because they have had less time to form cross-year bonds, and so are more assortative. The nose is created because the nodes that are most central within their year experience the least tension.

The three low assortativity universities have very different tension and elevation scores. It can be seen that higher tension appears to create a fuzzier, less clearly defined groups within each year. Low elevation creates a single 'nose' for the whole cone, but higher elevation starts separating out, forming a nose for each year. It is impractical to

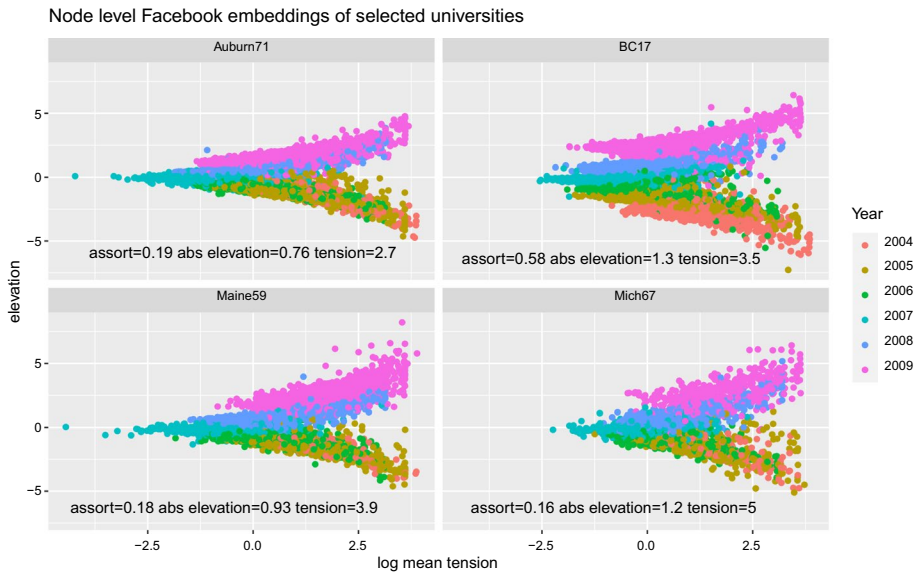

Fig. 7 The node level embedding shows that the tension and elevation distribution of nodes by type are distinct and related to the overall network tension and elevation; this is not possible to explore with assortativity

understand the differences of the universities by looking at the scatter plots of thousands of students across 100 universities. The elevation and node tension are thus aggregated at university level, and all 100 universities are plotted in Fig. 8. The points are coloured by assortativity. It can be seen that while there is a positive relationship between tension and elevation, there is, in fact, a negative relationship between the tension and elevation dimensions and the value of assortativity. Analysing the distribution of the tension, elevation and assortativity scores for all 100 universities, it is found that the data are normally distributed. Creating a linear regression on assortativity using tension and elevation as independent variables provides an $R^2 = 0.82$, where the coefficients are significant to $p < 0.001$. Creating linear models using either tension or elevation provides $R^2 = -0.006$ and 0.429, respectively. The other embedding methods did not produce insightful embeddings with regard to assortativity; the linear model of node2vec had an $R^2 = 0.36$, although it got a 20 point bump if squared terms were used, the other embedding methods had very low R^2 values. SDNE was not included in the analysis due to its poor performance on low dimensions in the previous section. In addition, LLE failed to converge for any of the Facebook networks at such low dimensions.

Next, the ability of the embeddings to be able to predict the k nearest-neighbour nodes is tested against a baseline of graph adjacent voting. The model is an effective predictor of year with high values of kappa (59% when k is 9), and balanced accuracy (77% when k is 9) indicating that the model predicts above the naive baseline value. However, the graph adjacency model outperforms the knn using SETSe by between 7 and 10% in terms of accuracy; kappa and f1 score and up to 15% on balanced accuracy. It should be noted that high levels of performance are expected as the model is predicting on the data it was embedded with. Despite this caveat and the poor performance against the adjacency voting model, the results show that the embeddings produced are meaningful, even on large and complex networks, although the actual graph structure is lost.

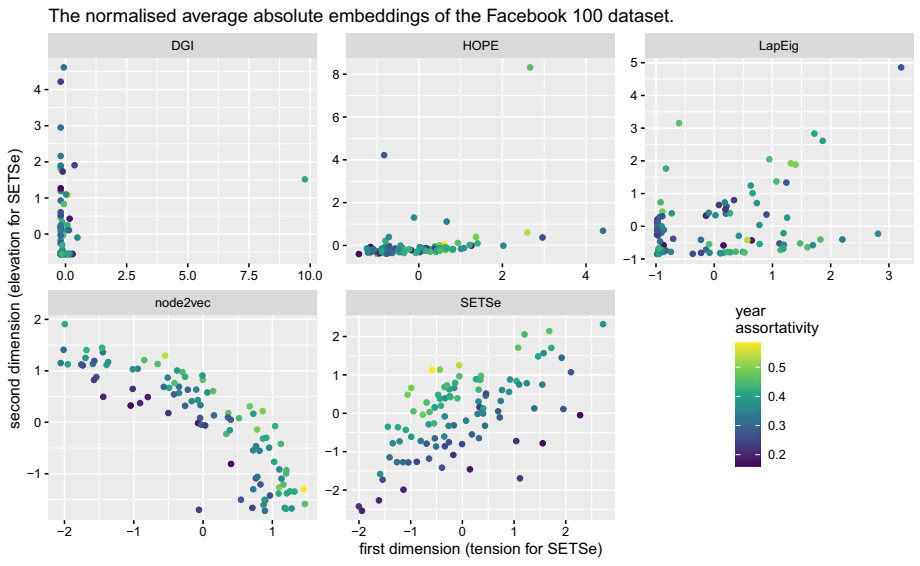

Fig. 8 There is a clear relationship between the SETSe embeddings and year assortativity; however, the pattern is dependent on elevation and tension. SETSe finds large differences between networks with very similar assortativity

Figure 6 showed that SETSe could uncover the hidden structure of the network; however, the question is can SETSe do that on a complex real-world dataset, and how does it compare to the other embedding methods? Using the same year embeddings for the Facebook 100 dataset, the k nearest neighbours were used to predict student type. The results show that the accuracy of the student-type embeddings averaged across all 100 universities. SETSe outperforms the naive rate of student type 2 over student type 1 at all values of k. SETSe's performance is also quite stable for all values of k, accuracy is around 71%, balanced accuracy is 55%, kappa is very low at 12% and the f1 score is around 79%. The low balanced accuracy is not significantly higher than 0.5, and the low kappa score signifies that the results could simply be due to chance. However, as shown in Fig. 9, the model comprehensively beats the nearest-neighbour voting method in all metrics for almost all values of k. SETSe also generally outperforms the other embedding methods, although Laplacian eigenvectors have a similar performance. Note that the f1 score is not reliable in this case as there are a significant number of occasions where type 1 students are not predicted at all, resulting in perfect prediction. This is because the f1 score is dependent on the class labelling; balanced accuracy and Cohen's kappa avoid this issue.

Complexity

The time taken to embed the Facebook graphs is plotted in Fig. 10. The top right panel of the figure shows that the iteration time complexity is linear ($\mathcal{O}(|\mathrm{E}|)$). This is fast for spring embedders, which generally have complexity $\mathcal{O}(|\mathrm{V}^2|)$ (Trenti and Hut 2008); it is also faster than FADE (Quigley and Eades 2001) which uses the Barnes–Hutt algorithm (Barnes and Hut 1986) to run in $\mathcal{O}(|\mathrm{E} + \mathrm{V} \log \mathrm{V}|)$. The time taken to reach convergence is shown in the right panel. The convergence complexity shows heteroscedasticity and is

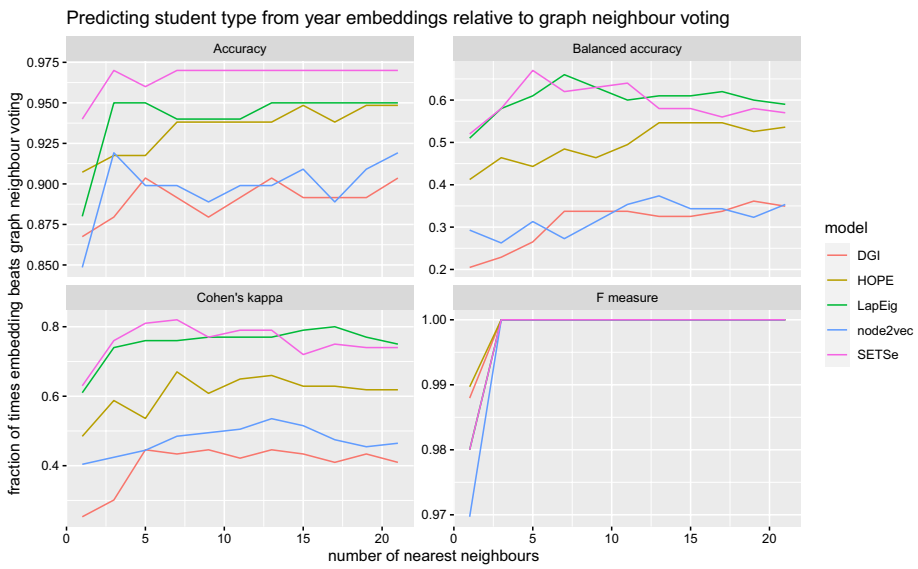

Fig. 9 Comparing embedding methods and the naive baseline. SETSe is generally the method with the best performance across the metrics, although is outperformed by Laplacian eigenvectors in some cases. SETSe beats the baseline more than 50% of the time for all values of k. Note for ease of viewing, the y-axis is truncated for each metric

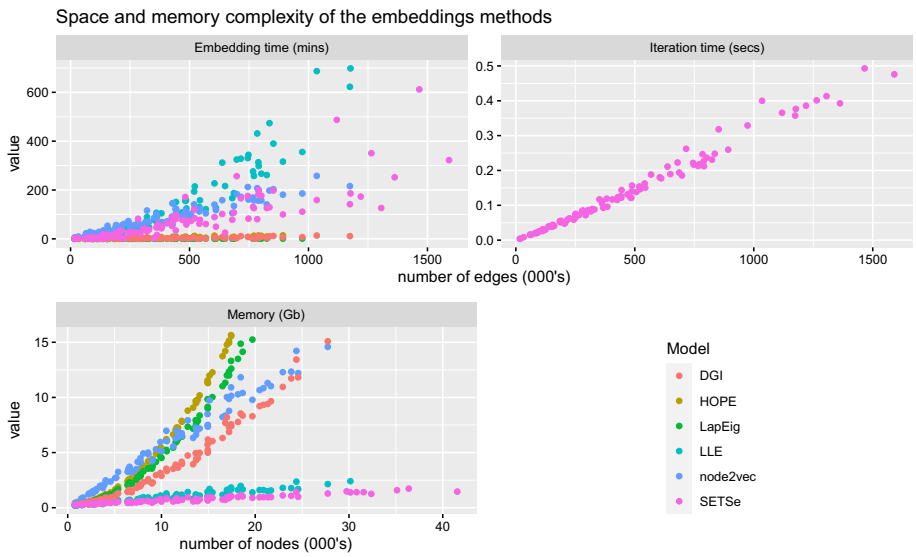

Fig. 10 The per-iteration complexity is linear for SETSe; however, the complexity of the total time to convergence is closer to quadratic. Memory complexity is linear and substantially lower than in the other methods

closer to running in quadratic time ($\mathcal{O}(|\mathrm{E}|^2)$). This is slower than DGI (Wu et al. 2020), LLE, Laplacian, Eigenmaps and HOPE, which have a complexity of $\mathcal{O}(|\mathrm{E}|)$, as well as node2vec ($\mathcal{O}(|\mathrm{V}|)$) and SDNE ($\mathcal{O}(|\mathrm{V}||\mathrm{E}|)$) for two-dimensional embeddings. The difference between the theoretical and empirical run times may be related to the implementation. It should be noted that SETSe's time to convergence is highly dependent on the network topology, using the bi-connected component method a tree network is solved in linear time; see the Additional file 1: Appendix for details on the bi-connected component method. The bottom left panel of Fig. 10 shows the space complexity of the algorithms measured in maximum memory used during the embedding process. SETSe has linear space complexity relative to the number of edges/nodes and lower memory use than all the other methods (in the figure, nodes are shown as the other methods are node dependent).

Discussion

SETSe acts as a hybrid between the advanced techniques of the graph embedders used for analysis and the intuitive simplicity of the spring embedders used for graph drawing. It does this by projecting the network onto an $n+1$-dimensional manifold, using node attribute as a force. This is different from traditional spring embedders where all nodes exert an equal force (Kamada and Kawai 1989; Fruchterman and Reingold 1991; EADES 1984). As such, unlike the graph embedding algorithms that were used to benchmark performance in this paper, SETSe cannot be said to 'learn' the properties of the network. Instead, similar to the other spring embedders, SETSe finds an equilibrium position wherein all forces are balanced. SETSe also distinguishes itself from many of the graph embedders and spring embedders by being entirely deterministic. In addition, SETSe provides an intuitive method to include node attribute data, these data act as the

'force' experienced by the nodes and acts orthogonally to the graph space. The SETSe algorithm is fast within each iteration, running in $\mathcal{O}(|E|)$ linear time. The time to convergence is not linear and appears to be closer to $\mathcal{O}(|E|^2)$, which is slower than most machine learning embedders. However, it is also linear in space complexity and appears to use substantially less memory than the comparison algorithms.

When separating Peel's quintet, only SETSe managed to find a successful two-dimensional representation of the networks. Two points should be made about this: the other techniques produce a two-dimensional representation of each node, whilst SETSe produces a one-dimensional representation of each node and a two-dimensional representation of each edge (although strain and tension are identical in this case). Another point is that the other graph embedding algorithms can embed the graph in any number of dimensions; typically, the graph would be embedded in higher-dimensional space then visualised in two dimensions by embedding the nodes a second time using some other data reduction method (Van Der Maaten and Hinton 2008; Pearson 1901). These two points illustrate the fundamentally different approach to embedding that SETSe takes as it is able to natively project high-dimensional data in meaningful low-dimensional space without needing a secondary embedding method. The explicit edge embedding also allows for flexibility when it comes to projection choices, which is something other embedding methods lack. The goal of typical graph embedders is to minimise the distance between nodes according to some measure of similarity. In contrast, SETSe does not try to optimise the meaning in the data; instead, it maps the attributes and edge weights of the network to a new space (the manifold), and in doing so reveals properties of the network.

When embedded in eight dimensions, most of the other graph embedders are able to linearly separate the sub-classes of Peel's quintet; this finding is similar to that of Goyal and Ferrara (2018). However, this leads to new problems such as how many dimensions should be used? What dimension reduction algorithm should then be chosen to reduce the dimensions again for plotting purposes, and how can the results be interpreted? In the case of using the output of the embedding for a model, dimensional parsimony is paramount. Being able to model the dependent variable in two dimensions is much more desirable than getting the same results with 16 dimensions. This is not to say that SETSe is always better than the other methods tested here. In particular, for applications where node similarity is the most important feature, or a large number of dimensions is advantageous, it would almost certainly be outperformed. Such a situation is the Facebook data, which would almost certainly benefit from a large number of embedded dimensions provided by more advanced methods such as those described in Grover and Leskovec (2016), Ou et al. (2016), or the hyperbolic algorithm described by Nickel and Kiela (2017). Unlike most machine learning graph embedders, SETSe is unable to perform link prediction. Being bound by physics also means SETSe cannot be easily tuned to target specific goals. However, while most graph embedders are designed to express a similarity chosen by the designer and parametrised by the user, SETSe in contrast is broadly goal agnostic. Instead, it simply allows the graph to express in a different way what was already there and provides insight into the underlying data in the process. In some cases, SETSe could be used in the preprocessing stage of the more sophisticated graph embedders, providing edge and attribute data to support similarity optimisation.

Conclusions

The SETSe algorithm is an unsupervised spring embedding method for graphs that uses a physics model to combine network topology with node and edge attributes. SETSe projects the graph onto an $n+1$-dimensional manifold, where n is the number of variables, and the final dimension is the graph space. The strain, elevation and tension values produced by the embedding process provide insight into the original data and can reveal structure that was not available when the embedding was produced.

Although SETSe could be classed as an unsupervised learning algorithm, SETSe does not learn any properties of the system or attempt to optimise similarity. As such, it can be considered as an auxiliary of, and a counterweight to, machine learning techniques. This fundamental difference is important as although the value of machine learning in current research progress cannot be overstated, the field of machine learning and artificial intelligence has been criticised as focusing too much on certain technologies, particularly deep learning (Klinger et al. 2020). In addition, often, the more sophisticated a technique, the more subjective choices are required, in development and in parametrisation.

This paper showed that SETSe outperforms several popular graph embedding algorithms, on tasks of network classification and node classification in low-dimensional space. SETSe also provides distinctions between networks that appear to be similar or identical when using the popular assortativity metric.

As was mentioned in the introduction, the SETSe algorithm can be applied to a variety of different problems such as power grid robustness, conflict on social networks, organisational culture as well as more graph-theoretic issues such as the graph isomorphism problem and chemoinformatics. These areas are the focus of further research.

Although spring embedders lost popularity with the rise of machine learning, SETSe shows that there is still a role for unsupervised physics models in modern data analysis. The spring has bounced back.

Supplementary information

Supplementary information accompanies this paper at https://doi.org/10.1007/s41109-020-00329-4.

Additional file

Additional file 1: Appendix. The Appendix contains details on the auto-SETSe and bi-connectedSETSe algorithms used to make network convergence easier.

Acknowledgements
I would like to thank Connor Galbraith and Patrick De Mars for their thoughtful and patient advice at all stages of this project; Dr. Ellen Webborn for her insightful and thorough feedback on the manuscript; and my supervisors Dr. Elsa Arcaute and Dr Aidan O'Sullivan for giving me the space to pursue this idea. I acknowledge use of the UCL Myriad High Performance Computing Facility (Myriad@UCL), and associated support services, in the completion of this work.

Authors' contributions
Not applicable

Funding
This work was funded by the EPSRC International Doctoral Scholars - IDS Grant (EP/N509577/1). The author declares that no outside body impacted the contents of this study.

Availability of data and materials
In addition an R package has been created, `rSETSe`, which can be used to create SETSe embeddings. The package can be installed from https://github.com/JonnoB/rSETSe. The Peel's Quintet networks (Peel et al. 2018), can be generated

from from the `generate_peels_network` function in the rSETSe package. The facebook data is available from Traud et al. (2012).

Competing interests
The authors declare that they have no competing interests

Received: 30 July 2020 Accepted: 22 October 2020
Published online: 04 November 2020

References

Aarseth SJ (2003) The N-body problem. In: Gravitational n-body simulations: tools and algorithms. Cambridge monographs on mathematical physics. Cambridge University Press, Cambridge, pp 1–17. https://doi.org/10.1017/CBO9780511535246.002
Anscombe FJ (1973) Graphs in statistical analysis. Am Stat 27(1):17–21. https://doi.org/10.2307/2682899
Barnes J, Hut P (1986) A hierarchical O(N log N) force-calculation algorithm. Nature 324(6096):446–449. https://doi.org/10.1038/324446a0
Belkin M, Niyogi P (2003) Laplacian eigenmaps for dimensionality reduction and data representation. Neural Comput 15(6):1373–1396. https://doi.org/10.1162/089976603321780317
Blondel VD, Guillaume J-L, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. J Stat Mech Theory Exp 2008(10):10008. https://doi.org/10.1088/1742-5468/2008/10/p10008
Cao S, Lu W, Xu Q (2016) Deep neural networks for learning graph representations. In: Thirtieth AAAI conference on artificial intelligence. https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12423 Accessed 2020-05-11
Chen H, Soni U, Lu Y, Maciejewski R, Kobourov S (2018) Same stats, different graphs. In: Biedl T, Kerren A (eds) Graph drawing and network visualization. Lecture notes in computer science. Springer, Cham, pp 463–477. https://doi.org/10.1007/978-3-030-04414-5_33
Clauset A, Newman MEJ, Moore C (2004) Finding community structure in very large networks. Phys Rev E. https://doi.org/10.1103/physreve.70.066111
Csardi G, Nepusz T (2006) The igraph software package for complex network research. Int J Complex Syst 1695
Data61 C (2018) StellarGraph Machine Learning Library. GitHub. Publication Title: GitHub Repository. https://github.com/stellargraph/stellargraph
Eades P (1984) A heuristic for graph drawing. Congressus Numerantium 42:149–160
Fey M, Lenssen JE (2019) Fast graph representation learning with PyTorch geometric. arXiv:1903.02428 [cs, stat]. arXiv:1903.02428. Accessed 29 May 2020
Frick A, Ludwig A, Mehldau H (1995) A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration). In: Tamassia R, Tollis IG (eds) Graph drawing. Lecture notes in computer science. Springer, Berlin, pp 388–403. https://doi.org/10.1007/3-540-58950-3_393
Fruchterman TMJ, Reingold EM (1991) Graph drawing by force-directed placement. Softw Pract Exp 21(11):1129–1164. https://doi.org/10.1002/spe.4380211102
Goyal P, Ferrara E (2018) Graph embedding techniques, applications, and performance: a survey. Knowl Based Syst 151:78–94. https://doi.org/10.1016/j.knosys.2018.03.022
Grover A, Leskovec J (2016) node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. KDD '16, pp 855–864. Association for Computing Machinery, San Francisco, California, USA. https://doi.org/10.1145/2939672.2939754. Accessed 11 May 2020
Gutiérrez-Gómez L, Delvenne J-C (2019) Unsupervised network embeddings with node identity awareness. Appl Netw Sci 4(1):1–21. https://doi.org/10.1007/s41109-019-0197-1
Kamada T, Kawai S (1989) An algorithm for drawing general undirected graphs. Inf Process Lett 31(1):7–15. https://doi.org/10.1016/0020-0190(89)90102-6
Kipf TN, Welling M (2016) Semi-supervised classification with graph convolutional networks
Klinger J, Mateos-Garcia J, Stathoulopoulos K (2020) A narrowing of AI research? arXiv:2009.10385 [cs]. arXiv: 2009.10385 . Accessed 30 Sep 2020
Kobourov SG (2013) Force-directed drawing algorithms. In: Tamassia R (ed) Handbook of graph drawing and visualization. CRC Press, Boca Raton, pp 383–408
Koren Y (2005) Drawing graphs by eigenvectors: theory and practice. Comput Math Appl 49(11):1867–1888. https://doi.org/10.1016/j.camwa.2004.08.015
Krzywinski M, Birol I, Jones SJ, Marra MA (2012) Hive plots-rational approach to visualizing networks. Brief Bioinform 13(5):627–644. https://doi.org/10.1093/bib/bbr069
Matejka J, Fitzmaurice G (2017) Same stats, different graphs: generating datasets with varied appearance and identical statistics through simulated annealing. In: Proceedings of the 2017 CHI conference on human factors in computing systems. CHI '17, pp 1290–1294. Association for Computing Machinery, Denver, Colorado, USA. https://doi.org/10.1145/3025453.3025912. Accessed 07 May 2020
Narayanan A, Chandramohan M, Venkatesan R, Chen L, Liu Y, Jaiswal S (2017) graph2vec: learning distributed representations of graphs. arXiv:1707.05005 [cs]. arXiv: 1707.05005. Accessed 23 Sept 2020
Nickel M, Kiela D (2017) Poincaré embeddings for learning hierarchical representations. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) Advances in neural information processing systems, vol 30, pp 6338–6347. Curran Associates, Inc. http://papers.nips.cc/paper/7213-poincare-embeddings-for-learning-hierarchical-representations.pdf. Accessed 21 Sep 2020
Ou M, Cui P, Pei J, Zhang Z, Zhu W (2016) Asymmetric transitivity preserving graph embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. KDD '16, pp 1105–1114.

Association for Computing Machinery, San Francisco, California, USA. https://doi.org/10.1145/2939672.2939751. Accessed 11 May 2020
Pearson K (1901) LIII. On lines and planes of closest fit to systems of points in space. https://doi.org/10.1080/14786440109462720. Accessed 29 May 2020
Peel L, Delvenne J-C, Lambiotte R (2018) Multiscale mixing patterns in networks. Proc Nat Acad Sci 115(16):4057–4062. https://doi.org/10.1073/pnas.1713019115.
Perozzi B, Al-Rfou R, Skiena S (2014) DeepWalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '14, pp 701–710. Association for Computing Machinery, New York, New York, USA. https://doi.org/10.1145/2623330.2623732. Accessed 11 May 2020
Pons P, Latapy M (2006) Computing communities in large networks using random walks. J Gr Algorithms Appl 10(2):191–218. https://doi.org/10.7155/jgaa.00124
Quigley A, Eades P (2001) FADE: graph drawing, clustering, and visual abstraction. In: Marks J (ed) Graph drawing. Lecture notes in computer science. Springer, Berlin, pp 197–210
Revell LJ, Schliep K, Valderrama E, Richardson JE (2018) Graphs in phylogenetic comparative analysis: Anscombe's quartet revisited. Methods Ecol Evol 9(10):2145–2154. https://doi.org/10.1111/2041-210X.13067
Roweis ST, Saul LK (2000) Nonlinear dimensionality reduction by locally linear embedding. Science 290(5500):2323–2326. https://doi.org/10.1126/science.290.5500.2323
Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G (2009) The graph neural network model. IEEE Trans Neural Netw 20(1):61–80. https://doi.org/10.1109/TNN.2008.2005605
Seo Y, Defferrard M, Vandergheynst P, Bresson X (2018) Structured sequence modeling with graph convolutional recurrent networks. In: Cheng L, Leung ACS, Ozawa S (eds) Neural information processing. Lecture notes in computer science. Springer, Cham, pp 362–373. https://doi.org/10.1007/978-3-030-04167-0_33
Springel V, White SDM, Jenkins A, Frenk CS, Yoshida N, Gao L, Navarro J, Thacker R, Croton D, Helly J, Peacock JA, Cole S, Thomas P, Couchman H, Evrard A, Colberg J, Pearce F (2005) Simulations of the formation, evolution and clustering of galaxies and quasars. Nature 435(7042):629. https://doi.org/10.1038/nature03597
Traud AL, Mucha PJ, Porter MA (2012) Social structure of Facebook networks. Physica A 391(16):4165–4180. https://doi.org/10.1016/j.physa.2011.12.021
Trenti M, Hut P (2008) N-body simulations (gravitational). Scholarpedia 3(5):3930. https://doi.org/10.4249/scholarpedia.3930
Tutte WT (1963) How to draw a graph. Proc Lond Math Soc 13(1):743–767. https://doi.org/10.1112/plms/s3-13.1.743
Van Der Maaten LJP, Hinton GE (2008) Visualizing high-dimensional data using t-sne. J Mach Learn Res. https://doi.org/10.1007/s10479-011-0841-3
Velič ković P, Fedus W, Hamilton WL, Liò P, Bengio Y, Hjelm RD (2018) Deep graph infomax. arXiv:1809.10341 [cs, math, stat]. arXiv: 1809.10341. Accessed 05 Oct 2020
Wang X, Zhang Y, Shi C (2019) Hyperbolic heterogeneous information network embedding. In: Proceedings of the AAAI conference on artificial intelligence, vol 33, no. 01, pp 5337–5344. https://doi.org/10.1609/aaai.v33i01.33015337. Number: 01. Accessed 21 Sep 2020
Wang D, Cui P, Zhu W (2016) Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 1225–1234. ACM, San Francisco, California, USA. https://doi.org/10.1145/2939672.2939753
Wu Z, Pan S, Chen F, Long G, Zhang C, Yu PS (2020) A comprehensive survey on graph neural networks. IEEE Trans Neural Netw Learn Syst. https://doi.org/10.1109/TNNLS.2020.2978386

Publisher's Note