SoftwareX 27 (2024) 101818

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

DTCNS : A python toolbox for digital twin-oriented complex networked systems

Jiaqi Wen *, Bogdan Gabrys, Katarzyna Musial

Complex Adaptive Systems Lab, Data Science Institute, University of Technology Sydney, PO Box 123. Broadway, Sydney, Australia

ARTICLE INFO

Keywords:
Complex networked systems
Digital twins
Network dynamics
Heterogeneous nodes attributes and features

ABSTRACT

Digital Twin-Oriented Complex Networked System (DT-CNS) is a modelling paradigm aiming at the representation and analysis of real-world networked systems. DT-CNS models network and the dynamics of and on the networks, including network growth dynamics and the dynamic process on networks. DT-CNS can be built and extended with increasing complexity towards the ultimate goal, a Digital Twin (DT) of reality. DT-CNS, as it approaches a DT, faithfully represents the temporal changes in network and processes, together with their interrelations, based on real-time information acquisition and feedback with reality. To build a DT-CNS with increasing complexity, one of the possible development paths is to increase the heterogeneity of nodes' features (characteristics of each specific node) and their preferences to create relationships while allowing the networks to evolve with the preference changes under the influence of dynamic process. To achieve that, we created a novel open-source Python tool named DTCNS to implement functionalities including feature representation, network growth based on heterogeneous node features and preferences to create relationships based on features, and the spreading process on networks. The source code, documentation and the full descriptions of the API are available at https://github.com/UTS-CASLab/DT-CNS.

Code metadata

Current code version	v1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00206
Permanent link to Reproducible Capsule	For example: https://codeocean.com/capsule/2328783/tree
Legal Code License	GPL-3.
Code versioning system used	none
Software code languages, tools, and services used	python
Compilation requirements, operating environments & dependencies	Python$\leq$ 3.6, numpy $\leq$ 1.14.6, pandas $\leq$ 0.25, networkx $\leq$ 2.8
If available Link to developer documentation/manual	https://github.com/UTS-CASLab/DT-CNS/blob/main/G1documents.ipynb
Support email for questions	jiaqi.wen@uts.edu.au

1. Motivation and significance

The DTCNS toolbox has been developed by the researchers within the Complex Adaptive Systems laboratory at the University of Technology Sydney. It models Digital Twin-Oriented Complex Networked Systems (DT-CNSs) with focus on heterogeneous node features and interaction rules.

Complex Networked Systems (CNSs) have been modelled with increasing complexity levels towards an accurate reflection of reality by introducing more heterogeneous network information and modelling more complex network dynamics [1,2]. Meanwhile, the DTs can provide an accurate reflection and, sometimes, an extension of reality as they represent historical events and simulate what-if scenarios that have never happened before [3]. Current research explores state-of-the-art DTs in network science field like Internet of Things (IoT) [4–6] and the supply chain management [7]. However, none of these studies focus on developing a CNS model towards a DT. The convergence between the goal of CNS modelling and the characteristics of a DT paradigm emerges as a new research focus, Digital Twin-Oriented Modelling of

* Corresponding author.
E-mail addresses: jiaqi.wen@uts.edu.au (Jiaqi Wen), Bogdan.Gabrys@uts.edu.au (Bogdan Gabrys), Katarzyna.Musial-Gabrys@uts.edu.au (Katarzyna Musial).

https://doi.org/10.1016/j.softx.2024.101818
Received 14 April 2024; Received in revised form 15 June 2024; Accepted 4 July 2024
Available online 24 July 2024

Table 1
Generations of DT-CNSs.

Generation	Requirements					Brief description
	Network	Process	Interrelation	Real-time information flow		
				One-way	Two-way	
1	Static	Static	×	×	×	Dynamic process on static networks
2a	Evolving	Static	×	×	×	Evolving dynamic process on static networks
2b	Static	Evolving	×	×	×	Dynamic process on evolving networks
3	Evolving	Evolving	✓	×	×	Evolving dynamic processes on evolving networks with interrelations between them
4	Temporal	Temporal	✓	✓	×	Temporal dynamic processes on temporal networks with interrelations between them and the acquisition of real time information
5	Temporal	Temporal	✓	✓	✓	Temporal dynamic processes on temporal networks with interrelations between them, as well as the real time two-way feedback between the reality and the CNSs, enabling an idealised state required by a DT

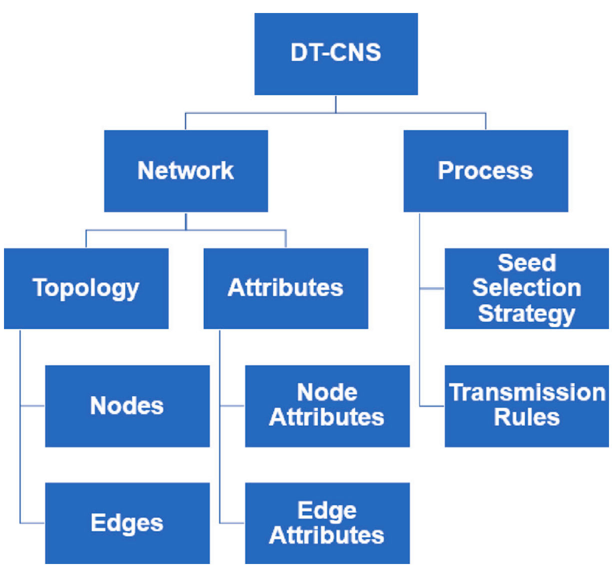

Fig. 1. The DT-CNS structure, including (i) network composed of network topology (nodes and edges) and attributes (for nodes and edges) and (ii) dynamic process on networks, driven by the seed selection strategy and transmission rules.

Complex Networked Systems [8]. In this space, a DT can be understood as an ultimate goal of CNS representation and modelling, which has not been realised yet but can be steadily approached by building and extending a DT-CNS towards a DT.

DT-CNSs aim at the faithful representation of networks and dynamics of and over networks, including the network dynamics and the dynamic process on the networks [1]. In a DT-CNS (see Fig. 1), the network can be composed of nodes and their connections (edges), together with their corresponding attributes (a.k.a. features). The network dynamics drive the edge formation and evolution in a network. The dynamic process on the networks are governed by the seed selection strategy (nodes from which the spread starts) and the transmissibility (infection rate given an exposure to spread).

DT-CNSs can be built and developed towards the ultimate goal of a DT with progressing complexity levels by introducing more heterogeneous network information and dynamics [1,8]. There are five generations of DT-CNSs, ranging from generation 1 to 5, that are characterised by an increasing complexity level (Fig. 2) and can achieved by fulfilling the requirements presented in Table 1.

As reviewed in our previous work [1], currently researched CNS dynamics generally fall in the generation 1 category by modelling the spreading process on static networks, such as the epidemic spread on social networks or trade risk spread on trade flow networks. Some studies focus on the evolving networks with network topology changes captured in snapshots over time (Generation 2b). Few reviewed studies focus on the evolving dynamic processes with parameter changes or consider interrelations between networks and processes (in generation 2a and generation 3). None of the studies investigates generation 4 and generation 5 models. The development path of a CNS towards a DT remains a research gap, which, among others, requires a Python tool to implement the DT-Oriented modelling tasks, including initialising a DT-CNS model and extending the DT-CNS towards higher complexity levels.

Therefore, inspired by the existing social network simulator built on heterogeneous node features and interaction rules [9], we decided to increase the heterogeneity of nodes' features and their preferences to create relationships while allowing the DT-CNSs to evolve with the preference changes under the impact of dynamic process. Our previous studies realise the conceptual ideas by building an extendable DT-CNS based on heterogeneous node features and connection preferences [8]. Moreover, we improve the expressive power of uncertain features in DT-CNSs by introducing fuzzy representation principles [10]. We further introduces changes in connection preferences to model evolving network structures under the impact of dynamic processes [11]. To implement these idea, we build an extendable Python tool, DTCNS, to enable the representation and analysis of networks and dynamic processes on networks in DT-CNSs. The proposed Python tool (i) generates networks based on heterogeneous node features and interaction rules, and (ii) simulates dynamic processes based on seed selection strategy and transmission rules [8]. The interaction rules cover the representation principles of uncertain features [10] and characterise nodes' connection preferences [8]. The seed selection strategy identifies the first infectious nodes and allows the propagation of spread based on a set of transmission rules concerning the infection rate and recovery rate.

Section 2 describes the architecture, functionalities and characteristics of software; Section 3 provides illustrative examples of building

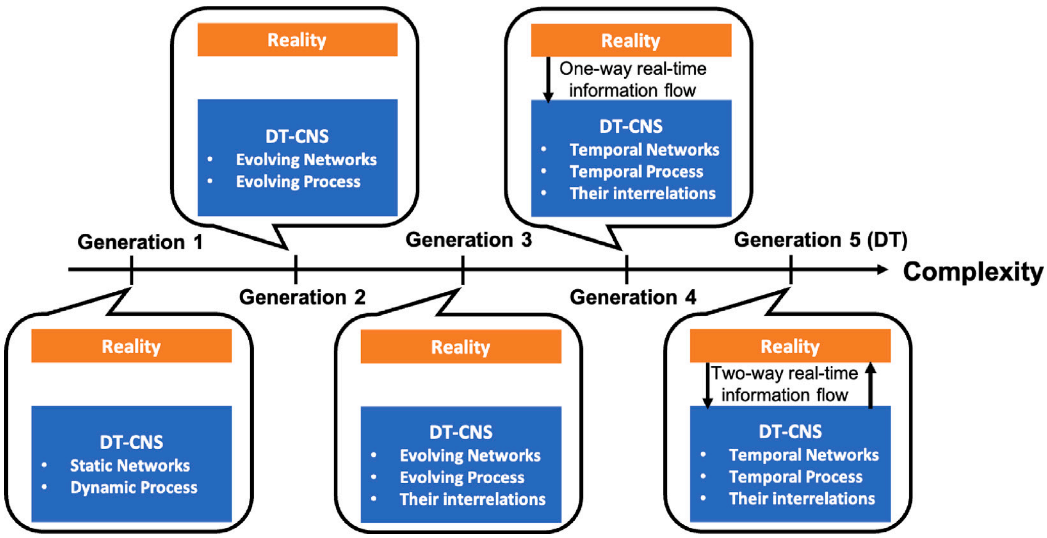

Fig. 2. The generations of DT-CNSs that progress with increasing complexity levels towards a DT.

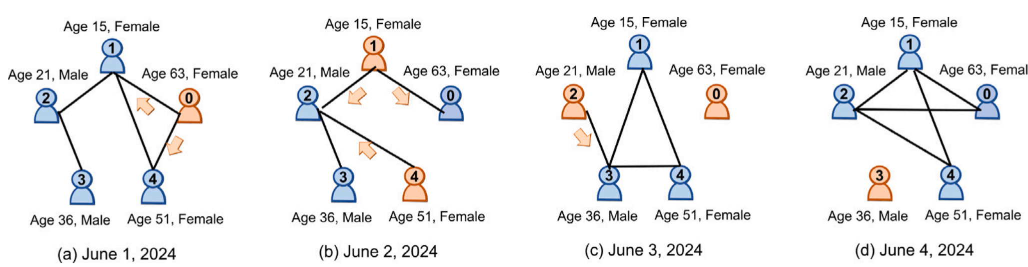

Fig. 3. An example of the Complex Networked Systems in Social Network Space.

a DT-CNS for social networked systems using the 2 toolbox; Section 4 discusses the software impact; Section 5 concludes this study and gives an outlook in DT-CNS extensions and applications.

2. Software description

DTCNS is a Python open source toolbox building Digital Twin-Oriented Complex Network Systems. The project has started in 2021 by Ms. Jiaqi Wen, Prof. Bogdan Gabrys[1] and Prof. Katarzyna Musial.[2]

The DT-CNSs are composed of networks and dynamics of and on the networks. The networks can be represented as set of nodes connected with each other via edges where both nodes and edges can have attributes. The DT-CNS dynamics can be either considered as: (i) dynamic processes on the networks, which involve spreading phenomena including epidemic processes and information spreading processes, or (ii) dynamic networks with evolving structures and attributes (a.k.a. features). For example, the DT-CNS of a social networked system can be composed of an dynamic social network and the epidemic spreading process that propagates on the social networks through social contacts (See Fig. 3).

2.1. Software architecture

The DTCNS tool is developed with the `networkx` library and generates a graph, formatted as a `graphml` object, and a health status vector indicating the health status of each node with boolean values. This tool models the network growth based on nodes' heterogeneous features (a.k.a. attributes) and the interaction rules that characterise (i) the nodes' representation of features and (ii) the nodes' preferences for features. In addition, the DTCNS tool models the spreading phenomenon in DT-CNSs based on (i) seed selection strategy and (ii) transmission rules concerning the infection rate and recovery rate of a spread (Fig. 4).

In Fig. 4, the construction of a network model in DT-CNS starts with the initialisation of feature representation principles [10] and feature preference principles [8]. We represent specific features with their standardised crisp values within $[0, 1]$. We represent uncertain features with fuzzy sets, named as a fuzzy representation principle. In addition, we initialise nodes' preferences for connecting with others based on the preferential attachment principles and homophily principles. Based on node features and the initialised rules related to feature representation and preference, the proposed DTCNS tool further enables us to calculate the node pair scores, rank preferred node pairs and connect the higher ranking node pairs to generate a static network.

To initialise a process model for the spreading process on networks with a DTCNS tool, we need to set up a seed selection strategy to

[1] https://profiles.uts.edu.au/Bogdan.Gabrys
[2] https://profiles.uts.edu.au/Katarzyna.Musial-Gabrys

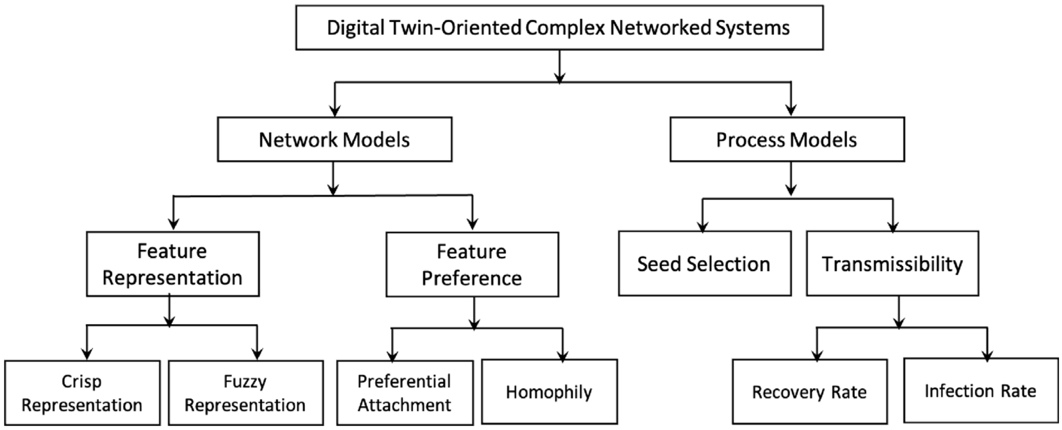

Fig. 4. The architecture of the DTCNS package.

determine the first infectious nodes, which may allow further propagation of the spread to their neighbours in the network. The spreading phenomenon on the networks is modelled by classic epidemiologic models such as the Susceptible–Infected (SI) model and its extensions, including the Susceptible–Infected–Susceptible (SIS) model [12], Susceptible–Infected–Recovered (SIR) model [13]. Therefore, we also need to initialise the transmissibility of spread, including recovery rate and infection rate. The infection rate determines the probability of infection given exposure to the spread. The recovery rate determines the probability of recovery after getting infected. Based on the generated network by the proposed DTCNS tool and the initialised parameters, including the seed and the transmissibility, the DTCNS tool further simulates a dynamic process that takes place on the networks overtime and, meanwhile, generates a dynamic vector to record the health status of each node.

2.2. Software functionalities

The DTCNS tool aims to build a DT-CNS for the static networks and the dynamic process on the networks, based on heterogeneous node features and the related rules of interaction and process propagation. To initialise a DT-CNS (as presented in Section 2.1) and enable the network generation and process simulation in DT-CNS, this DTCNS tool covers the following modules.

- **Feature** module enables the crisp representation of specific numerical features and the fuzzy representation of uncertain numerical features.
- **SocialDNA** module initialises and reserves the node's preferences for features and feature differences.
- **Network** module scores and ranks node pairs based on the represented nodes' features and the corresponding preferences. It further connects node pairs and generates networks.
- **Process** module simulates the dynamic process that starts from the seed node (nodes) given a specific transmissibility, including infection rate and recovery rate.

3. Illustrative examples

In this section, we present an example of building a generation 1 DT-CNS with the DTCNS tool for the epidemic spreading process on a static social network. The modelling process includes feature representation, preference representation, network growth based on the represented features and preferences and the process propagation.

3.1. Installation and usage

The DTCNS tool can be installed through cloning the code from our GitHub repository,[3] executing the existing setup script and installing the prerequisite Python libraries.

In the below example, we simulate the interactions and the epidemic spread between five nodes based on their preference for connecting with others. To implement this task with DTCNS toolbox, we firstly import the following packages:

Listing 1: Importing the packages

```
>>> from DTCNS.Network import Network,SocialDNA
>>> from DTCNS.Feature import Feature
>>> from DTCNS.Process import Process
>>> import pandas as pd
>>> import numpy as np
```

3.2. Feature representation

In the toy example, we consider two node features, age and sex.

Listing 2: Initialising the age and sex features.

```
>>> # Initialise the age and sex features:
>>> AgeSexData = pd.DataFrame(columns = ["Age","Sex"])
>>> AgeSexData["Sex"] = [0,1,1,0,1] # 1 and 0 each represents males
and females.
>>> AgeSexData["Age"] = [15,21,36,51,63]
>>> # The feature of each node:
>>> AgeSexData
   Age  Sex
0   15    0
1   21    1
2   36    1
3   51    0
4   63    1
```

3.2.1. Crisp representation of age and sex

When the nodes know the specific age and sex values, we can represent the features based on crisp representation principles.

- **Feature.CrispRepresentation()** standardises the numerical feature values into the range $[0, 1]$ and stores the crisp representation of features (stored in Feature.feature) and feature differences (stored in **Feature.featureDifferenceDict**).

[3] https://github.com/UTS-CASLab/DT-CNS

The nodes are denoted as 0, 1, 2, 3, 4 and stored in a list. Given a crisp representation principle, we have a crisp representation of features for each node and a feature difference dictionary recording the feature difference representation between each node and any other node.

Listing 3: Crisp representation of age and sex.

```
>>> # Crisp Representation of Age and Sex
>>> AgeSexFeatures = Feature(AgeSexData)
>>> AgeSexFeatures.CrispRepresentation()
>>> # The nodes reserved in a list
>>> AgeSexFeatures.nodes
[0, 1, 2, 3, 4]
>>> # Corresponding feature representation for each node
>>> AgeSexFeatures.feature
      Age   Sex
0   0.0000  0.0
1   0.1250  1.0
2   0.4375  1.0
3   0.7500  0.0
4   1.0000  1.0
>>> # Feature difference representation between node 3 and others.
>>> AgeSexFeatures.featureDifferenceDict[3]
   Age Difference   Sex Difference
0          0.7500              0.0
1          0.6250              1.0
2          0.3125              1.0
3          0.0000              0.0
4          0.2500              1.0
```

3.2.2. Fuzzy representation of age

When the nodes are uncertain about the age and the age difference, we can represent the age feature based on fuzzy representation principles.

- **Feature.FuzzyRepresentation()** unfolds a single numerical feature (feature difference) into a series of membership values ranging between [0,1]. This function builds fuzzy sets for the feature and the feature difference based on Gaussian Membership functions. It requires initialising the number of fuzzy sets and the corresponding Gaussian Membership functions.

Listing 4: Initialising an age feature

```
>>> # Initialise an age feature
>>> AgeData = pd.DataFrame(columns=["Age"])
>>> AgeData["Age"] = [15,21,36,51,63]
```

We initialise three fuzzy sets to represent the ages around 15, 30 and 60. The sigma values are set as 5, controlling the spread of fuzzy sets.

Listing 5: Initialising fuzzy sets for fuzzy representation of the age feature given the preferential attachment principle.

```
>>> # Initialise the fuzzy sets for the age features:
>>> # The number of fuzzy sets for age:
>>> NumFuzzyPdict = {"Age":3}
>>> # The parameter set-ups for Gaussian membership functions:
>>> FuzzyPparams = pd.DataFrame(columns=["mu","sigma"])
>>> FuzzyPparams["mu"] = [15,30,60]
>>> FuzzyPparams["sigma"] =[5,5,5]
>>> FuzzyPparamsDict = {"Age":FuzzyPparams}
>>> # The parameter set-ups of each fuzzy set:
>>> FuzzyPparamsDict
{'Age':   mu  sigma
 0  15      5
 1  30      5
 2  60      5}
```

Similarly, we initialise four fuzzy sets to represent the age differences around 0 and 12

Listing 6: Initialising fuzzy sets for fuzzy representation of the age difference given the homophily principle.

```
>>> # Initialise the fuzzy sets for age differences:
>>> # The number of fuzzy sets for age differences:
>>> NumFuzzyHdict = {"Age":2}
>>> FuzzyHparams = pd.DataFrame(columns=["mu","sigma"])
>>> FuzzyHparams["mu"] = [0,12]
>>> FuzzyHparams["sigma"] =[5,5]
>>> FuzzyHparamsDict = {"Age":FuzzyHparams}
>>> # The parameter set-ups of each fuzzy set:
>>> FuzzyHparamsDict
{'Age':   mu  sigma
 0   0      5
 1  12      5}
```

We can represent uncertain age features and age differences based on the above set-ups of fuzzy sets (three fuzzy sets for the age; two fuzzy sets for the age difference). The age feature is unfolded into three features, which indicate the memberships of an age value for the corresponding fuzzy set. Similarly, the age difference between the nodes can be unfolded into two membership values.

Listing 7: Fuzzy representation of age and age differences (e.g. the fuzzy age difference with the focal node 3)

```
>>> AgeFeatures = Feature(AgeData)
>>> AgeFeatures.FuzzyRepresentation(
    NumFuzzyPdict = NumFuzzyPdict,
    NumFuzzyHdict = NumFuzzyHdict,
    minValue = {"Age":0},#minimum feature values.
    maxValue = {"Age":70}, #maximum feature values.
    minDiffValue = {"Age":0}, #minimum feature differences with each
    focal node
    and other nodes
    maxDiffValue = {"Age":70}, #maximum feature differences with the
    focal node
    and others
    FuzzyPparamsDict = FuzzyPparamsDict, #dictionary with uncertain
    features as the key, including the parameter set-ups of fuzzy sets
    for uncertain features.
    FuzzyHparamsDict = FuzzyHparamsDict #dictionary with uncertain
    feature
    differences as the key, including the parameter set-ups of fuzzy
    sets for
    uncertain feature differences.
                    )

>>> # The unfolded membership values from an uncertain age feature,
corresponding to the three fuzzy sets for the representation of age:
>>> AgeFeatures.feature
```

```
    Age—Fuzzy0     Age—Fuzzy1     Age—Fuzzy2
0  1.000000e+00  1.110900e—02  2.576757e—18
1  4.867523e—01  1.978987e—01  6.148396e—14
2  1.477484e—04  4.867523e—01  9.929504e—06
3  5.534610e—12  1.477484e—04  1.978987e—01
4  9.720985e—21  3.475891e—10  8.352702e—01
>>> # The unfolded membership values from an uncertain age difference ,
with the focal node 3, corresponding to the two fuzzy sets for the
representation of age difference.
>>> AgeFeatures.featureDifferenceDict[3]
   Age Difference—Fuzzy0  Age Difference—Fuzzy1
0           5.534610e—12               0.000010
1           1.522998e—08               0.001534
2           1.110900e—02               0.835270
3           1.000000e+00               0.056135
4           5.613476e—02               1.000000
```

3.2.3. Feature update

We can update the feature representations with **Feature.FeatureUpdate()** function.

- **Feature.FeatureUpdate()** function updates a feature object with new features.

In this toy example, we replace the previous crisp representation of age with the fuzzy representation of age.

Listing 8: Updating the crisp representation of age with the fuzzy representation of age.

```
>>> # Feature update:
>>> AgeSexFeatures.FeatureUpdate(AgeFeatures, ReplaceFeatures=["Age"])
>>> # The updated features:
>>> AgeSexFeatures.feature
   Sex    Age—Fuzzy0    Age—Fuzzy1    Age—Fuzzy2
0  0.0  1.000000e+00  1.110900e—02  2.576757e—18
1  1.0  4.867523e—01  1.978987e—01  6.148396e—14
2  1.0  1.477484e—04  4.867523e—01  9.929504e—06
3  0.0  5.534610e—12  1.477484e—04  1.978987e—01
4  1.0  9.720985e—21  3.475891e—10  8.352702e—01
>>> # The updated feature differences with the focal node 2:
>>> AgeSexFeatures.featureDifferenceDict[2]
  Sex Difference  Age Difference—Fuzzy0  Age Difference—Fuzzy1
0            1.0           1.477484e—04               0.197899
1            0.0           1.110900e—02               0.835270
2            0.0           1.000000e+00               0.056135
3            1.0           1.110900e—02               0.835270
4            0.0           4.655716e—07               0.011109
```

3.3. Preference representation

Corresponding to the age and sex representations, we set up the related preferences and weights of preferences. These preference set-ups are formatted into a **SocialDNA** object.

- **SocialDNA()** module incorporates the preference for features (**pDNA**) and the preference for feature differences (**hDNA**), together with their respective weights of preference (**hWeight** and **pWeight**). hDNA and hWeight (**pDNA** and **pWeight**) are stored in data frames and take the same format as the features (feature differences). The preference **pDNA** and **hDNA** take trinary values at 1, 0 and −1, each indicating negative, neutral and positive preferences. The preference weights **hWeight** and **pWeight** take numerical values within $(0, 1]$.

In this toy example, we assume that all nodes share the same social preferences and preference weights. They have positive preferences for age and sex features and negative preferences for age and sex differences (a.k.a. positive preferences for the similar age and the similar sex). Meanwhile, the weights of preference are set to the maximum value at one.

Listing 9: Initialising preferences into a **SocialDNA** format.

```
>>> # Initialise the preferences for features:
>>> pDNA = pd.DataFrame(np.ones([AgeSexFeatures.numNodes,
AgeSexFeatures.numFeatures]),
columns = AgeSexFeatures.feature.columns)
>>> # Initialise the weights of preference for features:
>>> pWeight = pd.DataFrame(np.ones([AgeSexFeatures.numNodes,
AgeSexFeatures.numFeatures]),
columns = AgeSexFeatures.feature.columns)
>>> # Initialise the preferences for feature differences:
>>> hDNA = pd.DataFrame(—np.ones([AgeSexFeatures.numNodes,
AgeSexFeatures.numFeatureDifferences]),
columns = AgeSexFeatures.featureDifferenceDict[0].columns)
>>> # Initialise the weights of preference for feature differences:
>>> hWeight = pd.DataFrame(np.ones([AgeSexFeatures.numNodes,
AgeSexFeatures.numFeatureDifferences]),
columns =AgeSexFeatures.featureDifferenceDict[0].columns)
>>> # Format the preferences into a \textbf{SocialDNA} object:
>>> myDNA = SocialDNA(AgeSexFeatures,hDNA = hDNA,pDNA = pDNA, hWeight
= hWeight, pWeight = pWeight)
```

For example, the nodes' preferences and preference weights for feature differences can be represented as follows:

Listing 10: Nodes' preferences and preference weights for feature differences.

```
>>> # Represent the preferences for feature differences:
>>> myDNA.hDNA
  Sex Difference  Age Difference—Fuzzy0  Age Difference—Fuzzy1
0           —1.0       —1.0                    —1.0
1           —1.0       —1.0                    —1.0
2           —1.0       —1.0                    —1.0
3           —1.0       —1.0                    —1.0
4           —1.0       —1.0                    —1.0
>>> # Represent the weights of preferences for feature differences:
>>> myDNA.hWeight
  Sex Difference  Age Difference—Fuzzy0  Age Difference—Fuzzy1
0            1.0        1.0                     1.0
1            1.0        1.0                     1.0
2            1.0        1.0                     1.0
3            1.0        1.0                     1.0
4            1.0        1.0                     1.0
```

3.4. Network formation

We can simulate an undirected static network based on the feature representations and preferences with the **Network()**.

- **Network()** requires the essential information for network generation. In addition to features and preferences, we can set up an encounter rate (**encounterRate**) that describes the probability

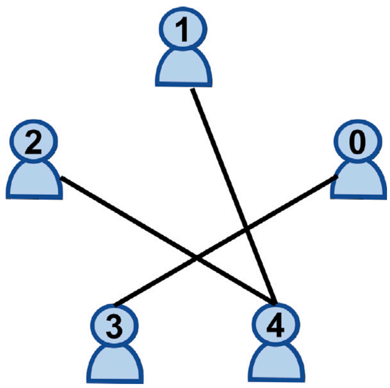

Fig. 5. The static social network generated by the **Network** module for the five nodes attributed with different age and sex features.

of encountering another node, the number of edges (**NumEdge**), an interaction cost (**InteractionCost**) between 0 and 1 that determines the edge number and the random interference that is generated from a normal distribution $N(0, \mathbf{randomInterference}^2)$. The random interference ranges in $(0, 0.1)$.
Please note that we can only set one of these variables, NumEdge and InteractionCost, to determine the edge number.

- **Network.Formation(randomseed)** function generates an undirected network snapshot. We can reproduce the same network by setting up the random seed **randomseed**.
- **Network.writeHistory(saveDir)** function saves the preferences and the graph to the path **saveDir**.

We assume an encounter rate at 0.80 and 3 edges to create a static network for the five nodes in this toy example. The random interference is 0.005, with minor impact on edge formation. The generated network is stored in the **myNetwork.Graph** object.

Listing 11: Network formation based on the feature representation and preferences.

```
>>> # Initialise the \textbf{Network} model based on features and
related preferences:
>>> myNetwork = Network(Feature = AgeSexFeatures,socialDNA = myDNA,
encounterRate = 0.8,
NumEdge =  3, InteractionCost = None, randomInterference = 0.005)
>>> # Generate a network based on the random seed 112:
>>> myNetwork.Formation(randomseed=112)
>>> # The edges in the simulated network:
>>> myNetwork.Graph.edges()
EdgeView([(0, 3), (1, 4), (2, 4)])
>>> # Save the network:
>>> myNetwork.writeHistory(saveDir="Data")
```

In the generated static social network (Fig. 5). Node denoted by 4, attributed with the oldest age 63 and a male feature, has the highest node degree at 2. This results from the nodes' features and preferences. The nodes in this network prefer similar sex features and male features and, meanwhile, prefer ages around 15, 30 and 60 and age differences around 0 and 12. Node denoted by 4 has a preferred male feature around two females and another two males. In addition, node 4 also has a preferred age around 60 around other nodes and a preferred age difference around 12 with node 3. These features and corresponding preferences finally lead to connections between the male node 4 and male nodes denoted by 1 and 2.

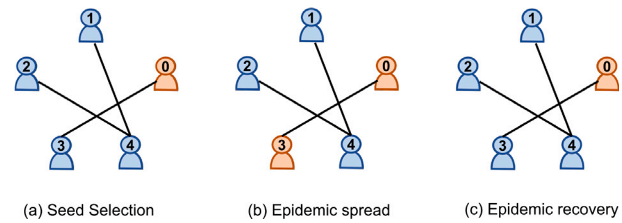

Fig. 6. The seed selection in the first time step and the epidemic spread and recovery in another two time steps with the **Process** module.

3.5. Process propagation

We simulate the dynamic process on networks with the **Process** module.

- **Process()** simulates the dynamic process on the networks. It is initialised with the node number **numNodes**, a set of seed nodes **infectedSet**, an infection rate **infectionRate** and a recovery rate **recoveryRate**.

In the toy example, we assume an epidemic spread on the simulated social network from the seed node 0 (included in the **infectedSet**). The infection rate and the recovery rate are initialised as 0.2. We set a random seed (its default value is 112) for the reproducible results and, in the simulations, allow the process to spread for one step and the nodes to recover for another step. The infected nodes are stored in a list named **infectedSet**.

Listing 12: Python example

```
>>> # Initialise the epidemic spreading process with the seed
selection and
the transmissibility concerning the infection rate and recovery rate:
>>> epiProcess = Process(numNodes=5,infectedSet={0},infectionRate=0.2,
recoveryRate=0.2)
>>> # Epidemic spread:
>>> epiProcess.spread(myNetwork.Graph,randomseed = 112,spreadStep = 1)
>>> #The infected node set after epidemic spread:
>>> epiProcess.infectedSet
{0, 3}
>>> #Epidemic recovery:
>>> epiProcess.recover(myNetwork.Graph,randomseed = 112,recoverStep
= 1)
>>> #The infected node set after epidemic recovery:
>>> epiProcess.infectedSet
{0}
>>> #To save the information related to the epidemic spreading
process:
>>> epiProcess.writeHistory(saveDir="Data",recoveryRecord=True,
spreadRecord=True)
```

In Fig. 6, we represent the healthy nodes and the infected nodes with the blue and the red colour. The epidemic spreading process starts from the seed node denoted by 0 and, in the next time step, infects its neighbour, the node denoted by 3, which gets recovered in a further time step. In this context, the python tool captures the rule that nodes with preferred features have more interactions and higher infection risks.

Overall, the toy example simulates static social networks and the epidemic spreading process on the static networks with DTCNS toolbox. The DT-CNS built in this example falls in the generation 1 category

Table 2
Development path of DT-CNS dynamics from generation 1 to generation 5 (the ultimate goal of a DT) with increasing complexity levels.

Generation	DT-CNSs		Reality
	Interaction rule	Transmission rule	Decision-making
1	Non-Evolutionary	Non-Evolutionary	×
2	Non-Evolutionary	Evolutionary	×
	Evolutionary	Non-Evolutionary	×
3	Co-Evolutionary		×
4	Co-Evolutionary and Real-time Updatable		×
5	Co-Evolutionary and Real-time Updatable		

(See Fig. 2 and Table 1) and requires further extensions towards higher generations based on the existing and future functionalities in DTCNS toolbox.

4. Impact

The DTCNS toolbox enables the construction of generation 1 DT-CNSs and benefits the research community due to its extendability and flexibility.

Regarding the extendability of the DTCNS toolbox, the generation 1 DT-CNSs can be extended to higher complexity generations by introducing changes to features and related rules, including feature representation and feature preferences. We provide one of the possible development paths to extend the DT-CNSs towards the DT of a real-world networked system though introducing (co-)evolutionary and real-time updatable interaction rules (feature representation and preferences) and transmission rules (seed selection and transmissibility) [8]. This development path progresses over five DT-CNS generations (See Fig. 2, Tables 1 and 2) and gradually enables the evolvability in dynamics, interconnections between evolutionary dynamics and the interplay between the DT-CNSs and the reality.

Due to the flexibility of the DTCNS toolbox in introducing node features and the related preferences, the DTCNS toolboxolbox can be employed in multiple disciplines by capturing various node characteristics and their connection preferences. It can model the social networked systems, where nodes have social contact based on interaction rules and, meanwhile, transmit the epidemic, opinion, influence or any other spread to neighbours. For example, in the DT-CNSs of a social network under an epidemic outbreak, we model the social network growth based on nodes' heterogeneous features (e.g. age, sex, height, popularity, etc.) and their interaction rules that characterise their preferences for nodes' features (e.g. preference for a similar age and a young age) [8,10].

5. Conclusion

The DTCNS toolbox is intended for researchers and practitioners as an easily accessible toolbox of DT-CNS models. This tool is implemented in Python, enabling the construction of a generation 1 DT-CNS focusing on the dynamic process on static networks, based on heterogeneous node features and related rules.

The DTCNS toolbox is currently in active development and holds the potential to simulate networked systems across various scales. In the future, we will introduce novel elements and new functionalities to DTCNS tool to more faithfully represent and model heterogeneous network components and their interrelated dynamics. We will also consecutively work on DT-CNSs of large, complex real-world networked systems, and expand their applications to multiple disciplines.

CRediT authorship contribution statement

Jiaqi Wen: Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Bogdan Gabrys:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization. **Katarzyna Musial:** Writing – review & editing, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgement

This work was supported by the Australian Research Council, Dynamics and Control of Complex Social Networks under Grant DP190101087.

References

[1] Wen J, Gabrys B, Musial K. Toward digital twin oriented modeling of complex networked systems and their dynamics: A comprehensive survey. IEEE Access 2022;10:66886–923. http://dx.doi.org/10.1109/ACCESS.2022.3184801.

[2] Wen J, Gabrys B, Musial K. Review and assessment of digital twin–oriented social network simulators. IEEE Access 2023;11:97503–21. http://dx.doi.org/10.1109/ACCESS.2023.3312129.

[3] Jans-Singh M, Leeming K, Choudhary R, Girolami M. Digital twin of an urban-integrated hydroponic farm. Data-Centric Eng 2020;1.

[4] Minerva R, Lee GM, Crespi N. Digital twin in the iot context: A survey on technical features, scenarios, and architectural models. Proc IEEE 2020;108(10):1785–824.

[5] Bellavista P, Giannelli C, Mamei M, Mendula M, Picone M. Application-driven network-aware digital twin management in industrial edge environments. IEEE Trans Ind Inf 2021;17(11):7791–801.

[6] Kirchhof JC, Michael J, Rumpe B, Varga S, Wortmann A. Model-driven digital twin construction: synthesizing the integration of cyber–physical systems with their information systems. In: Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems. 2020, p. 90–101.

[7] Kamble SS, Gunasekaran A, Parekh H, Mani V, Belhadi A, Sharma R. Digital twin for sustainable manufacturing supply chains: Current trends, future perspectives, and an implementation framework. Technol Forecast Soc Change 2022;176:121448.

[8] Wen J, Gabrys B, Musial K. Towards digital twin-oriented complex networked systems: Introducing heterogeneous node features and interaction rules. PLoS One 2024;19(1):e0296426.

[9] Ashraf AW-U, Budka M, Musial K. Simulation and augmentation of social networks for building deep learning models. 2019, arXiv preprint arXiv:1905.09087.

[10] Wen J, Gabrys B, Musial K. Heterogeneous feature representation for digital twin-oriented complex networked systems. 2023, arXiv preprint arXiv:2309.13229.

[11] Wen J, Gabrys B, Musial K. Evolutionary digital twin-oriented complex networked systems driven by node features and the mutation of feature preferences. PLoS One 2024;19(5):e0303571.

[12] Liu Q, Van Mieghem P. Burst of virus infection and a possibly largest epidemic threshold of non-markovian susceptible-infected-susceptible processes on networks. Phys Rev E 2018;97(2):022309.

[13] Tomé T, Ziff RM. Critical behavior of the susceptible-infected-recovered model on a square lattice. Phys Rev E 2010;82(5):051921.