GeometricFlux.jl：Flux 上的幾何深度學習

GeometricFlux.jl: Geometric Deep Learning on Flux

Speaker: Yueh-Hua Tu（杜岳華）

Why graph neural network?

Nowadays AI

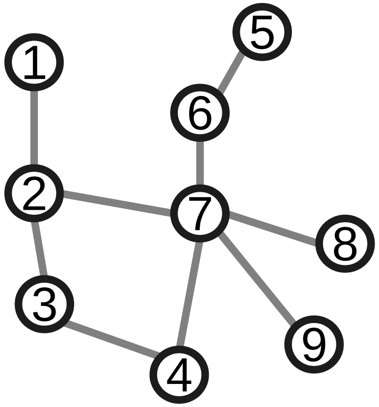
- Computer vision
- Natural language processing
- Speech
- Game, including cheese game, real-time strategy (RTS)

Many scientific data lies in non-Euclidean space

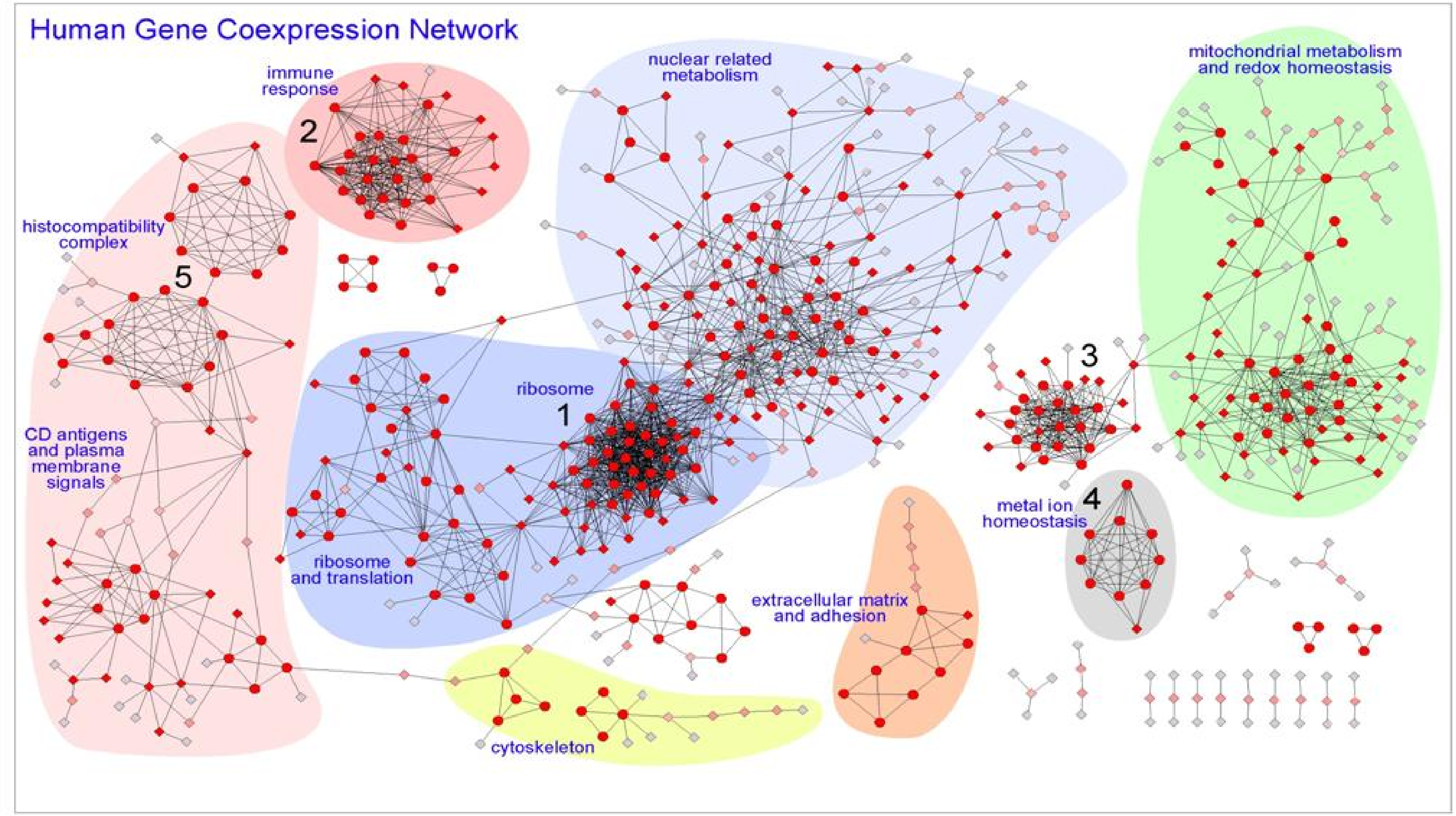
- Social networks in socail science
- Traffic networks
- Gene regulatory networks in biology
- Molecular structure in chemistry
- Knowledge graph
- 3D object surfaces in computer graphics

Graph

They can be represented as a graph structure.

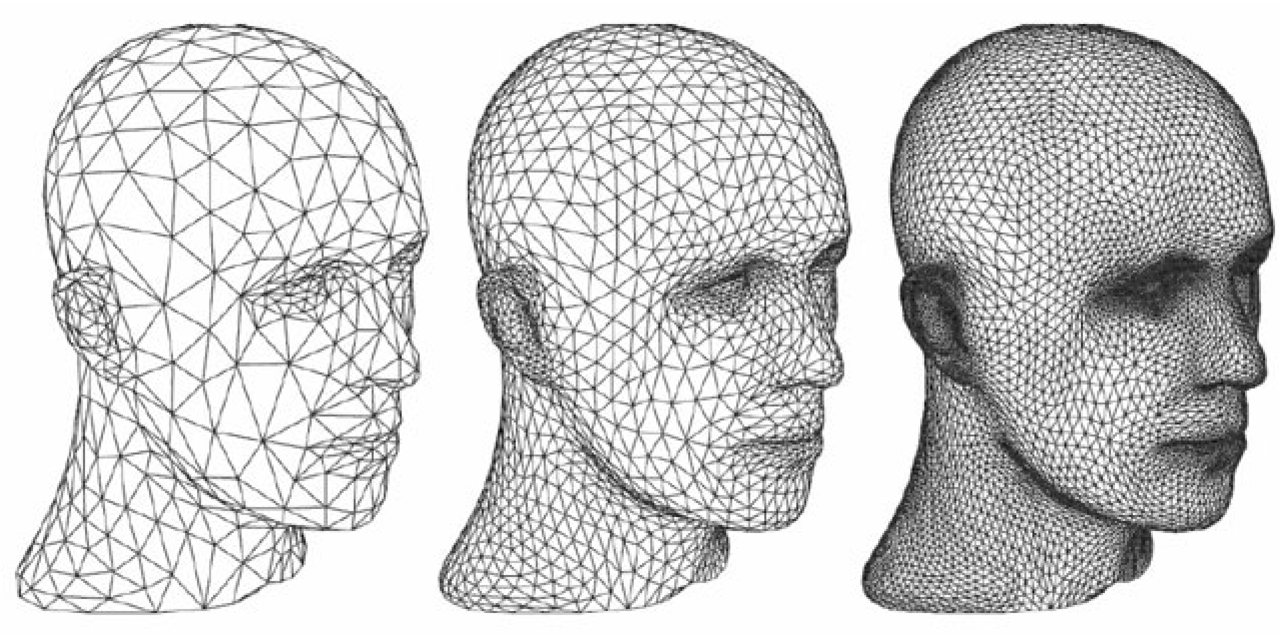


Human gene coexpression network


Human Gene Coexpression Network
immune response
2
nuclear related metabolism
mitochondrial metabolism and redox homeostasis
histocompatibility complex
5
3
ribosome
1
CD antigens and plasma membrane signals
metal ion homeostasis
4
ribosome and translation
extracellular matrix and adhesion
cytoskeleton

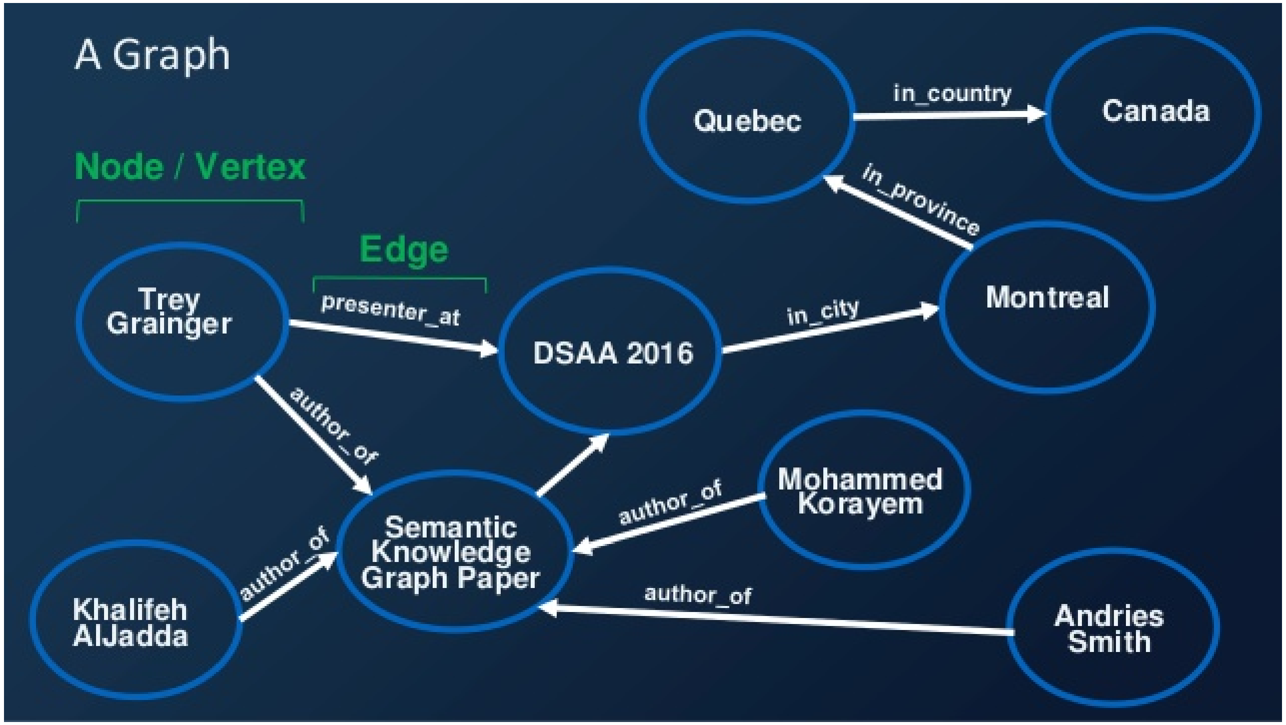
picture source (http://bioinfow.dep.usal.es/coexpression/)

Computer graphics



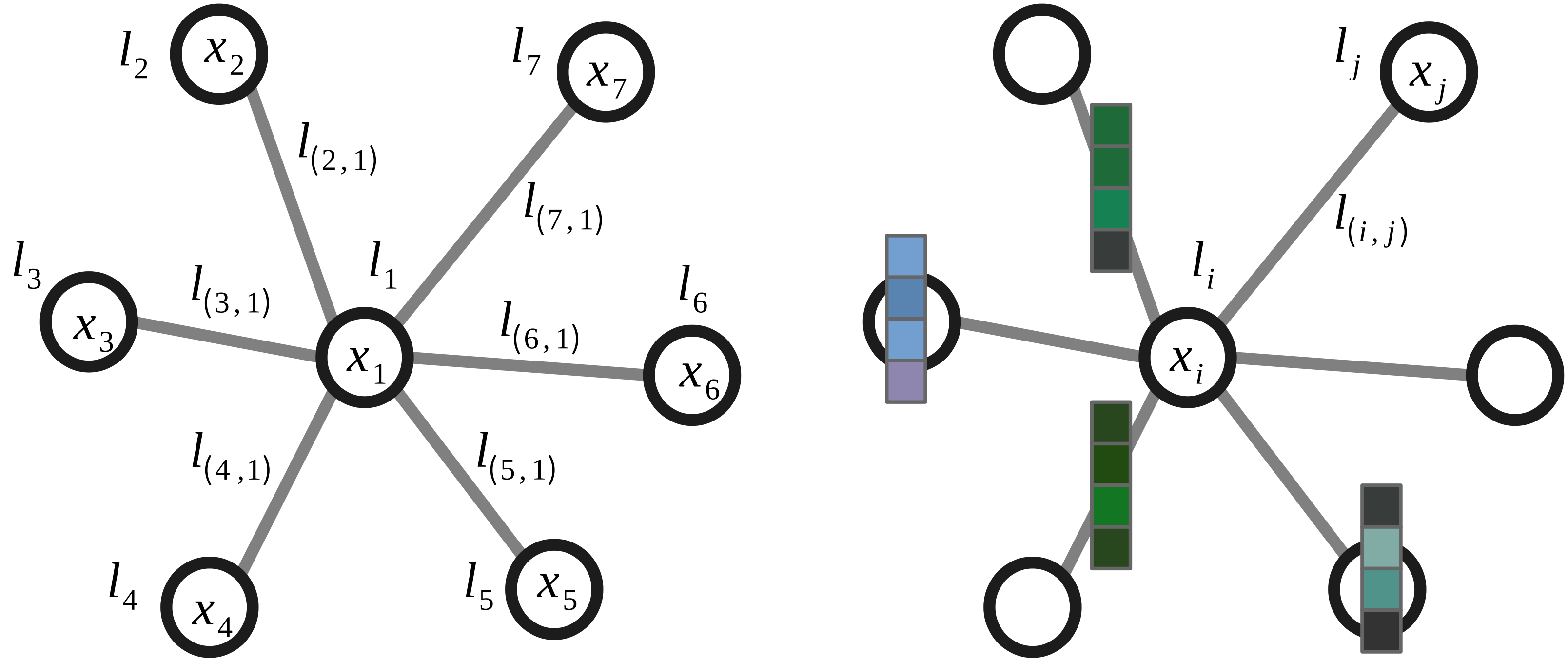
[picture source (https://computergraphics.stackexchange.com/questions/2018/what-is-tessellation-in-computer-graphics)](https://computergraphics.stackexchange.com/questions/2018/what-is-tessellation-in-computer-graphics)

Knowledge graph

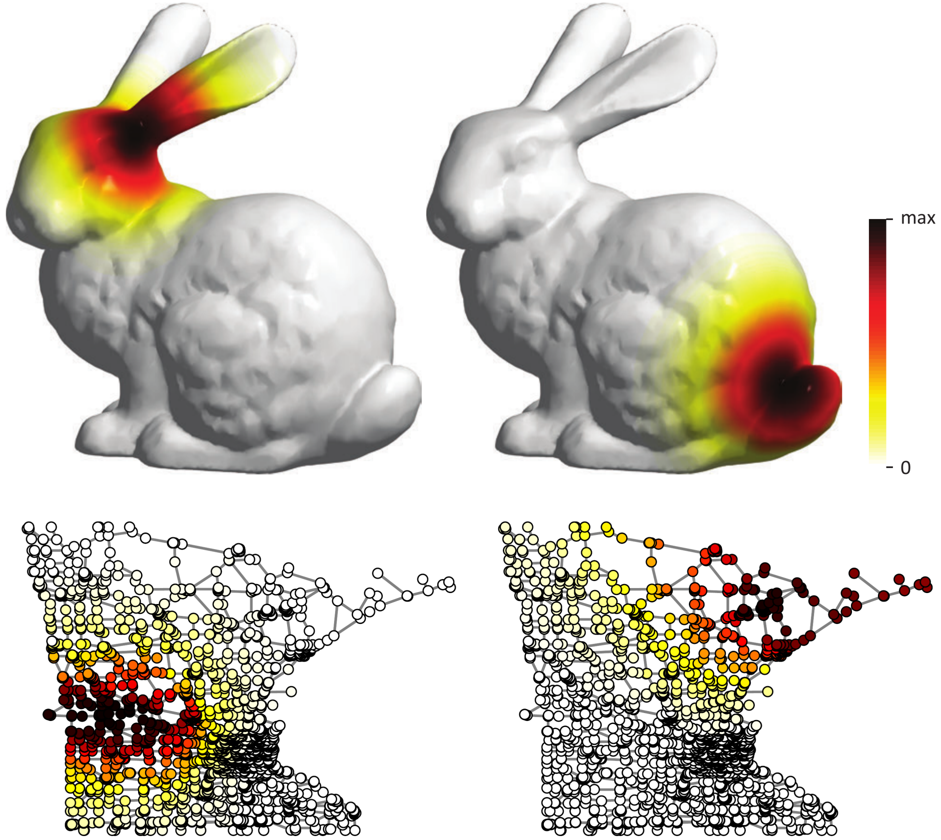


[picture source (https://www.slideshare.net/treygrainger/the-semantic-knowledge-graph)](https://www.slideshare.net/treygrainger/the-semantic-knowledge-graph)

Graphs are more representative than vectors



Graphs are discrete approximation to manifold



Geometric deep learning: going beyond Euclidean data (https://arxiv.org/abs/1611.08097)

Geometric Deep Learning

Graph + Deep Learning = Powerful

GeometricFlux

https://github.com/yuehhua/GeometricFlux.jl (https://github.com/yuehhua/GeometricFlux.jl)

Aims to

- extend **Flux** deep learning framework in Julia
- support of CUDA GPU with **CuArrays** (other interfaces are welcome)
- integrate with existing **JuliaGraphs** ecosystem

Graphs are usually sparse

I worked on SparseArrays and CuArrays.CUSPARSE

Layers

- Convolution layers
  - MessagePassing
  - GCNConv
  - GraphConv
  - ChebConv
  - GatedGraphConv
  - GATConv
  - EdgeConv
  - Meta (WIP)
- Pooling layers
  - GlobalPool (WIP)
  - TopKPool (WIP)
  - MaxPool (WIP)
  - MeanPool (WIP)
  - sum/sub/prod/div/max/min/mean pool
- Embedding layers
  - InnerProductDecoder

Models

- VGAE
- GAE

Compatible with layers in Flux

In [ ]:

```
## Model
model = Chain(GCNConv(g, num_features=>1000, relu),
              GCNConv(g, 1000=>500, relu),
              Dense(500, 7),
              softmax)
```

Use it as you use Flux

In [ ]:

```
## Loss
loss(x, y) = crossentropy(model(x), y)
accuracy(x, y) = mean(onecold(model(x)) .== onecold(y))

## Training
ps = Flux.params(model)
train_data = [(train_X, train_y)]
opt = ADAM(0.01)
evalcb() = @show(accuracy(train_X, train_y))

Flux.train!(loss, ps, train_data, opt, cb=throttle(evalcb, 10))
```

Construct layers from SimpleGraph/SimpleWeightedGraph

In [ ]:

```
g = SimpleGraph(5)
add_edge!(1, 2); add_edge!(3, 4)
GCNConv(g, num_features=>1000, relu)
```

Use Zygote (with CPU)

GeometricFlux with Zygote on GPU is not available. There are some issues to work on...

For GPU, I will getting Tracker work first.

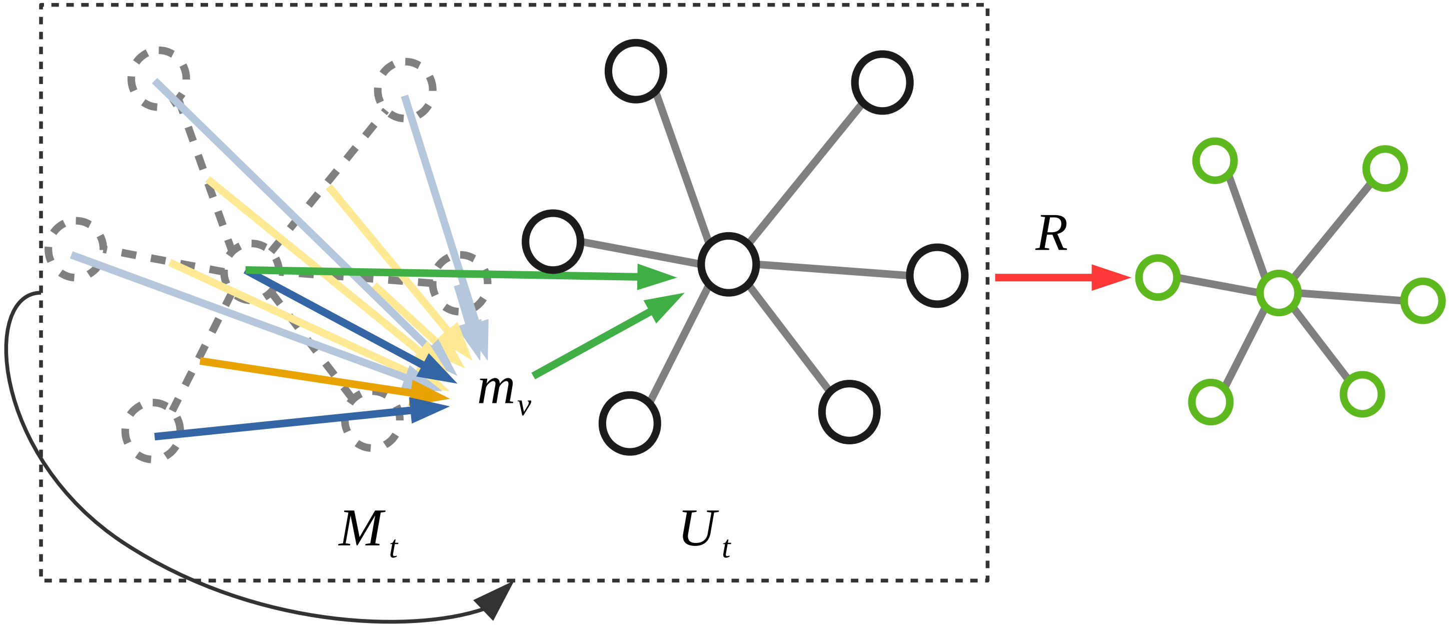
Use message passing scheme

$$m_v^{t+1} = \sum_{u \in N(v)} M_t(h_v^t, h_u^t, e_{uv})$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

h_v^t: vertex features, e_{uv}: edge features, $N(v)$: neighbors of vertex v

Message passing scheme



Performance

- Multithreaded scatter function
- Scatter function on GPU (WIP)

Some work to do

- Support CUDA/CUSPARSE
- Accept edge list dynamically
- More layers/models
- Datasets inetegration
- Threading on CPU (in v1.3!)
- Sparse array computation optimization

Thank you for attention

Pull request and issues are welcome.