Graph algorithms for very large graphs and their applications to bioinformatics and social network analysis

Pasquale Foggia
University of Salerno, Italy
pfoggia@unisa.it

MIVIA Lab
Intelligent Machines for Video, Image and Audio Analysis
MIVIA Lab

✧ 3 full professors
  ● Mario Vento, Francesco Tortorella, Pasquale Foggia

✧ 2 associate professors
  ● Gennaro Percannella, Pierluigi Ritrovato

✧ 3 researchers
  ● Alessia Saggese, Luca Greco, Vincenzo Carletti

✧ 1 post-doc

✧ 3 PhD students
MIVIA Lab

- Artificial vision
- Cognitive robotics
- Medical image analysis
- Structural Pattern Recognition and Graph-based Algorithms
  - Graph-based algorithms applied to genomics
(Attributed) Graphs

✦ Representation for *structured* information

- Nodes (aka vertices) representing atomic entities
- Edges representing relations between entities
- Nodes and edges can have *attributes* (aka *labels*) carrying additional information about entities and relations

![Graph Diagrams]
Obvious examples

H-C-OH
C=O
H-C-OH
H-C-OH

Start
Look for lost item

Did you find it?

Yes
No
Do you need it?

Customer
- CustomerID
- Name
- Mail
- Zip Code
- Address
- Telephone

Order
- OrderID
- CustomerID (FK)
- OrderDate
- TotalPrice

OrderDetail
- OrderID (FK)
- OrderDetailID
- ProductID (FK)
- Amount
- Price

Product
- ProductID
- Name
- Kind
- Price

Graph algorithms for very large graphs and their applications…
Less obvious examples

✦ Image segmentation
Less obvious examples

✧ Semantic data

- E.g. DBpedia stores information from Wikipedia as RDF graphs
Less obvious examples

- **Deep Learning optimization**

  - **Tensorflow** represents the DNN operations as a dataflow graph, then uses this graph to code the computation for the available CPUs/GPUs.
Less obvious examples

✧ ... we even tend to imagine graphs when we look at the night sky!
A common operation on graphs is **matching**:

- Find a correspondence between the nodes of the two graphs that preserves some interesting properties
- Different kinds of matching depending on the properties one wants to preserve
Isomorphism

✧ Find if two graphs have the same structure

1  2
3  4

a  b

a  c  b

c  d
Isomorphism

♦ Find if two graphs have the same structure
Subgraph isomorphism

✧ Find if a graph contains the other as a subgraph

\[ \begin{align*}
1 & \quad 2 \\
3 & \quad 1 \\
\end{align*} \]

\[ \begin{align*}
a & \quad b \\
c & \quad d \\
\end{align*} \]
Subgraph isomorphism

✧ Find if a graph contains the other as a subgraph
Maximum common subgraph

Find the largest subgraph common to the two graphs
Maximum common subgraph

Find the largest subgraph common to the two graphs
Maximum common subgraph

- MCS is related to the Maximum Clique problem: find the largest subgraph that is *fully connected*
Graph edit distance

- Find the “minimum cost” set of edit operations that makes the first graph isomorphic to the second
Graph edit distance

✦ Find the “minimum cost” set of edit operations that makes the first graph isomorphic to the second

Remove edge (3,4)
Add edge (4,2)
Add node 5
Add edge (2,5)
All the above problems except isomorphism have been proved to be NP-Complete.

For isomorphism there is a demonstration (pending verification) that the problem is QP (Quasi-Polynomial), with complexity:

\[ T(n) = 2^{O((\log n)^3)} \]
Large graphs

- Early applications of graph matching only worked with very small graphs
- e.g. my (modest) very first scientific paper!
Large graphs

✦ Initial generation of algorithms able to work with graphs ranging from tens of nodes (graph edit distance) to a hundred nodes (subgraph isomorphism)

✦ In the late ‘90s and early 2000s several works addressed the problem of large graphs
Subgraph isomorphism

✧ The most popular subgraph isomorphism algorithm was Ullmann’s (1976)
✧ Ullmann required a memory space that is cubic ($O(n^3)$) wrt to the size of the graphs
  ● Only suitable up to a few hundred nodes
In 1999 we proposed the VF algorithm, that requires $O(n^2)$ space

Subgraph isomorphism

In 2004 we proposed the VF2 algorithm, that requires $O(n)$ space


Fig. 3. Matching times for the subgraphs of the MAP-1 image (a) and of the ENGINE-2 image (b).
In 2007, Riesen, Neuhaus and Bunke proposed a method for approximating the GED using the Linear Assignment Problem, that can be solved in $O(n^3)$ time.

In 2012, we used approximate GED for analyzing the dynamics of socio-economic networks


- Applied to the network induced by mutual participation in the board of directors of Italian firms in the industrial, financial and publishing sector from 2007 to 2011, to detect structural changes
Graph edit distance

✧ In 2017, we proposed a better but slower approximation using the Quadratic Assignment Problem

Graph edit distance

✧ LAP vs QAP

Time

Error
Large enough?

Although the above mentioned algorithms made possible to use larger graphs than before, new applications raised the bar regarding what “large” means.
Examples: bioinformatics

馏 Protein Data Bank: world wide collection of information on proteins
馏 Protein chemical structures: 500 to 10000 nodes, very sparse (max degree 4)
馏 Protein contact maps: 150 to 800 nodes, but denser (average degree 20)
馏 Protein interaction networks:
馏 Yeast: ~3000 nodes, ~12000 edges
馏 Human: ~4600 nodes, ~86000 edges
Examples: social networks

✧ **Social circles in Facebook** (ego-Facebook, 2012)
  ✧ ~4000 nodes representing (anonymized) Fb users, with ~90000 undirected edges representing friendship

✧ **Wikipedia Admin elections** (wiki-Vote, 2010)
  ✧ ~7000 nodes representing (anonymized) Wikipedia users, with ~100000 directed edges representing votes for admin role
Several algorithms have been proposed in the 2010s, that try to improve over 2nd generation matching algorithms using better heuristics (making assumption on the graph properties)
Bonnici and Giugno proposed RI, a subgraph isomorphism algorithm suited to large and sparse graphs in bioinformatics.

VF3

✧ We proposed VF3, an evolution of VF2 better suited to large and dense graphs

✧ V. Carletti, P. Foggia, A. Saggese and M. Vento, Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3, in IEEE Trans. on PAMI, vol. 40, no. 4, pp. 804-818, 2018
Node reordering

✦ Both RI and VF3 use the idea of reordering the nodes before starting the matching so as to detect as early as possible if a tentative matching is going to fail

- The ordering criteria are different
- VF3 also has additional heuristics for dense graphs
Node reordering

Starting here will take several steps before discovering that the small graph is not contained in the large one.
Node reordering

✦ Starting here will take several steps before discovering that the small graph is not contained in the large one.

![Graph Diagram]

1. Starting at node 1, the algorithm may proceed to node 2, then to node 3, and so on.
2. The small graph consists of nodes a, b, c, d, e, f, g, and h.
3. The large graph consists of nodes 1, 2, 3, 4, 5, and a, b, c, d, e, f, g, h.
4. The algorithm must explore the entire large graph to determine if the small graph is contained within it.
Node reordering

Starting here will take several steps before discovering that the small graph is not contained in the large one.
Node reordering

✦ Starting here will take several steps before discovering that the small graph is not contained in the large one.
Node reordering

Starting here will take several steps before discovering that the small graph is not contained in the large one.
Node reordering

✧ Instead, starting here will immediately discover that no node can be associated to “3”
RI and VF3 performance

Protein structures
RI and VF3 performance

Dense random graphs
TurboISO

✧ Avoids exploring equivalent permutations of nodes by using auxiliary data structures

TurboISO

✧ Can be very efficient in matching very small subgraphs with very large graphs

Human Protein Interaction Network (~4600 nodes)
Parallel approach

- Given the wide availability of multiprocessor systems, several works have proposed parallelization as a mean for reducing the matching time.
- Usually using a shared memory architecture.
VF3P

- We have developed a parallel version of VF3, called VF3P (presented at GbR2019)
  - We cannot divide the graph among the processors (data parallelism)
  - So we divide the parts of the solution space (using a State-Space-Representation) to be explored (task parallelism)
  - Finding the right granularity
  - Reducing the communication overhead
VF3P

Our preliminary results show a speedup that is close to the number of processors when the graphs get large

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Target Size</th>
<th>Speed-up</th>
<th>VF3P</th>
<th>VF3P_{LS}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2 core</td>
<td>4 core</td>
<td>8 core</td>
</tr>
<tr>
<td>Uniform</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0.76</td>
<td>0.75</td>
<td>0.70</td>
<td>0.99</td>
</tr>
<tr>
<td>2000</td>
<td>1.59</td>
<td>2.34</td>
<td>3.42</td>
<td>1.55</td>
</tr>
<tr>
<td>4000</td>
<td>1.56</td>
<td>2.96</td>
<td>5.16</td>
<td>1.69</td>
</tr>
<tr>
<td>10000</td>
<td>1.79</td>
<td>3.44</td>
<td>6.36</td>
<td>1.82</td>
</tr>
<tr>
<td>Non-Uniform</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0.70</td>
<td>0.75</td>
<td>0.61</td>
<td>1.06</td>
</tr>
<tr>
<td>2000</td>
<td>1.54</td>
<td>2.16</td>
<td>2.98</td>
<td>1.51</td>
</tr>
<tr>
<td>4000</td>
<td>1.70</td>
<td>2.98</td>
<td>4.77</td>
<td>1.72</td>
</tr>
<tr>
<td>10000</td>
<td>1.77</td>
<td>3.37</td>
<td>6.11</td>
<td>1.80</td>
</tr>
<tr>
<td>Uniform</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1.72</td>
<td>2.63</td>
<td>3.46</td>
<td>1.79</td>
</tr>
<tr>
<td>2000</td>
<td>1.46</td>
<td>3.00</td>
<td>5.19</td>
<td>1.54</td>
</tr>
<tr>
<td>4000</td>
<td>1.78</td>
<td>3.36</td>
<td>6.05</td>
<td>1.81</td>
</tr>
<tr>
<td>6000</td>
<td>1.85</td>
<td>3.56</td>
<td>6.68</td>
<td>1.88</td>
</tr>
<tr>
<td>8000</td>
<td>1.88</td>
<td>3.64</td>
<td>6.82</td>
<td>1.90</td>
</tr>
<tr>
<td>Non-Uniform</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1.65</td>
<td>2.53</td>
<td>2.93</td>
<td>1.77</td>
</tr>
<tr>
<td>2000</td>
<td>1.59</td>
<td>2.93</td>
<td>4.91</td>
<td>1.66</td>
</tr>
<tr>
<td>4000</td>
<td>1.76</td>
<td>3.27</td>
<td>5.69</td>
<td>1.77</td>
</tr>
<tr>
<td>8000</td>
<td>1.86</td>
<td>3.60</td>
<td>6.74</td>
<td>1.88</td>
</tr>
</tbody>
</table>

\( \eta = 0.2 \)

\( \eta = 0.3 \)
Future challenges

✧ The algorithms we have seen allow us to work with graphs with tens of thousands of nodes

✧ Still, we have new applications yielding much larger graphs
Future challenges

✦ The Stanford Large Networks dataset collections (SNAP, http://snap.stanford.edu) includes very large graphs from social networks

✦ LiveJournal members: almost 5 millions nodes, 69 millions edges

✦ Friendster on line gaming: ~65 millions nodes, 1.8 billions edges
Future challenges

✦ A new trend in bioinformatics is the use of graph representations for genomic data

✦ Genome graph: represents the genetic information of a population, describing the common parts and the individual variations as paths within the graph

✦ A genome graph for 1000 human subjects: about 100 millions nodes, 3-400 millions edges
Graph summarization

✧ One approach for working with such large graphs is Graph Summarization

✧ Convert groups of nodes into nodes for obtaining a smaller graph with the same overall structure
Graph summarization

✦ A recent paper proposes a nice theoretical result about summarization


✦ Basically, the RL says that every graph large enough and dense enough is partitionable into a bounded number of regular bipartite graphs plus a small number of extra nodes and edges

✦ Does it work on practical applications? Can it be used to implement some form of hierarchical matching?
Subquadratic algorithms

✦ Another way to tackle these huge graphs is to replace information that is expensive to compute with approximations that can be computed in less than $O(n^2)$ time.

✦ Example: in Social Network Analysis, cliques can be replaced by $k$-cores as a way of finding groups of nodes with strong internal connections.
Subquadratic algorithms

- $k$-Cores can be computed in $O(k \times n)$ time.

- Can we do something similar for (some form of) graph matching?
Distributed computing

✦ Shared-memory parallelism has inherent limits on the scalability it can achieve
✦ Big Data applications often use other architectures, with distributed computing
  ✦ Example: the Map/Reduce programming model made popular by Google
Spark/GraphX

- The Apache distributed computing framework Spark includes an extension named GraphX offering APIs for implementing massively parallel algorithms on graph
- the programming model can be seen as an adaptation to graphs of Map/Reduce
- Facebook uses a similar approach for running some graph-based algorithms (e.g. for page ranking)
- Can graph matching (or a suitable surrogate of it) be adapted to such a paradigm?
GPU algorithms

✧ GPU accels make available tens to hundreds of teraflops at an affordable price

✧ we don’t have yet graph matching algorithms able to fully exploit this huge computational power (GPUs are best suited for data parallelism, while most parallel matching algorithms use task parallelism)

✧ Can graph matching be formulated in a data-parallel way?
Neural networks

✧ Deep neural network can approximate very complicated functions (e.g. object recognition). And what’s best is that they *learn* how to do it!

✧ with tensor GPU accelerators they can be quite fast

✧ Can we use deep learning to learn how to match graphs (even approximately)?
Neural networks

✧ Can we learn a graph similarity measure?
✧ Given two graphs, the network should output an approximation of their edit distance
✧ Most graph NN process one graph at a time
Can we learn how to search a pattern subgraph inside a larger target graph?

- YOLO detects and classifies smaller objects inside a large image...
Neural networks

✧ Can we learn a useful compact representation for graphs?
✧ a sort of graph autoencoder
✧ this representation can be used as the starting point for other operations (e.g. graph similarity)
✧ Several graph NN have already been proposed for node embedding
Conclusions

✧ In the last three decades we have increased the size of the graphs in our applications from a few nodes to tens of thousands of nodes and more

✧ We are now at a point where we cannot just evolve our algorithms, if we want to face the challenge coming from the huge graphs of Bioinformatics and Social Network Analysis
P. Foggia - Graph algorithms for very large graphs and their applications…
h t
a y o q u
n k u s e
s i o n ?

P. Foggia - Graph algorithms for very large graphs and their applications…