Cheetah: Fast Graph Kernel Tracking on Dynamic Graphs

Presenter: Liangyue Li
Joint work with
Hanghang Tong (ASU), Yanghua Xiao (Fudan), Wei Fan (Baidu)
Graphs are Everywhere

Collaboration Networks

US Power Grid

Bus Network

Brain Networks

Patient Networks

Hospital Networks
**Application 1: Web Mining**

Q: How similar are the two graphs?  
A: Graph Kernel

1. For each entity, construct a neighborhood graph by breadth-first search up to depth $k$

2. Apply graph kernel in kernel based learning methods

---

Q: How similar are the two graphs?
A: Graph Kernel

1. For each image, represent it as a segmentation graph
2. Apply graph kernel in kernel based learning methods

Q: How similar are the two graphs?
A: Graph Kernel

1. For each brain image, represent it as a graph
2. Apply graph kernel in kernel based learning methods
Intuitions:
1. Compare similarity of every pair of nodes from each graph
   — Eg: (1,2) vs (a, j) → less similar
   (1,5) vs (a,e) → more similar
2. Node pair similarity is measured by random walks
3. Two graphs are similar if they share many similar node pairs
RWR Graph Kernel — Formulation

Taking expectations instead of summing

\[
\text{Ker}(G_1, G_2) = \sum_k c^k q' x A^k x p x = q' x (I - c A x)^{-1} p x
\]

Computational challenge:

- \(A_x\) is of size \(n^2 \times n^2\)
- \(O(n^6)\) (Direct computation) or \(O(n^3)\) (Sylvester equation)

Time > 1h, n=3328

**Speed up — ARK**

**Idea:** perform low-rank approx on both graphs

\[
\text{Ker}(G_1, G_2) = (q_1' \otimes q_2')(I - cA_1' \otimes A_2')^{-1}(p_1 \otimes p_2)
\]

**Step 1:**

**Top-r low-rank approx:**

\[
U_1 \Lambda_1 V_1' \quad \quad U_2 \Lambda_2 V_2'
\]

**Step 2:**

**Matrix-inverse Lemma:**

\[
\text{Ker}(G_1, G_2) \approx (q_1' p_1)(q_2' p_2) + c(q_1' U_1 \otimes q_2' U_2) \tilde{\Lambda}(V_1' p_1 \otimes V_2' p_2)
\]

\[
\tilde{\Lambda} = ((\Lambda_1 \otimes \Lambda_2)^{-1} - c(V_1' \otimes V_2')(U_1 \otimes U_2))^{-1}
\]

- **Matrix of size** $r^2 \times r^2$, easy to inverse
- **Overall complexity:** $O(n^2 r^4 + mr + r^6)$

Time = 7.5s, n=3328

Can be reduced to $O(nr^2 + mr + r^6)$
Challenges

- ARK: Good for static graphs
- What if graphs are **evolving** over time

**Q:** How to track graph kernel efficiently?
Roadmap

▪ Motivation

▪ *Cheetah-D* for Directed Graphs

▪ Experimental Results

▪ Conclusion


**Cheetah-D: graph kernel tracking**

\[
\text{Ker}(G_1, G_2) = (q_1' \otimes q_2')(I - cA_1' \otimes A_2')^{-1}(p_1 \otimes p_2)
\]

top-r low-rank approx:

\[
U_1 \Lambda_1 V_1' \\
\Delta A_1
\]

\[
U_2 \Lambda_2 V_2'
\]

\[
\Delta A_2
\]

**Ideas:**

**Avoid:** re-computing low-rank approx [ARK]

**Goal:** track low-rank structure efficiently [Cheetah-D]

(SVD in this paper)
Step 0: Low rank approx on $\Delta A$

**Intuition:**

$A_0 \approx u_1 u_2 \times \Lambda_0 \times v_1 v_2$  
\[
\begin{bmatrix}
A_0
\end{bmatrix}
\approx
\begin{bmatrix}
u_1 & u_2
\end{bmatrix}
\times
\begin{bmatrix}
\Lambda_0
\end{bmatrix}
\times
\begin{bmatrix}
v_1 & v_2
\end{bmatrix}
\]

\[ (m = 5, r = 2) \]

$\Delta A \approx x_1 Y Z'$  
\[
\begin{bmatrix}
\Delta A
\end{bmatrix}
\approx
\begin{bmatrix}
x_1
\end{bmatrix}
\times
\begin{bmatrix}
Y
\end{bmatrix}
\times
\begin{bmatrix}
Z'
\end{bmatrix}
\]

\[ (m' = 2, r' = 1) \]

**Details:**

$A_0 \approx U_0 \Lambda_0 V_0'$  
\[
A_0 \approx
\begin{bmatrix}
U_0 & \Lambda_0 & V_0'
\end{bmatrix}
\]

$\Delta A = X Y Z'$  
\[
\Delta A =
\begin{bmatrix}
x
\end{bmatrix}
\times
\begin{bmatrix}
Y
\end{bmatrix}
\times
\begin{bmatrix}
Z'
\end{bmatrix}
\]

\[
A = A_0 + \Delta A
\]

\[
A =
\begin{bmatrix}
A_0 & 0
\end{bmatrix}
\begin{bmatrix}
U_0 & V_0'
\end{bmatrix}
\]

**Property:**

- SVD on $A_0$ takes: $O(mr + nr)$
- SVD on $\Delta A$ takes: $O(m'r' + nr') \ll O(mr + nr)$
  
  $[m' \ll m, r' \ll r]$
Step 1: Partial QR Decomposition

**Intuition:** 
**Case 1:** \( x_1 \in \text{span}(u_1, u_2) \)

**Details:** 
\[
\begin{bmatrix} U_0 & X \end{bmatrix} = U_0S
\]

**Property:**
- **Efficiency:** takes \( O(nr'^2) \) for \( r' \ll r \)
- **Effectiveness:** No extra error
Step 1: Partial QR Decomposition

**Intuition:**
Case 2: \( x_1 \notin \text{span}(u_1, u_2) \)

**Details:**
\[
\begin{bmatrix}
U_0 & X
\end{bmatrix} = \begin{bmatrix}
U_0 & q
\end{bmatrix} S
\]

**Property:**
- **Efficiency:** takes \( O(nr'^2) \) \( r' \ll r \)
- **Effectiveness:** No extra error

Similar Partial QR decomp on Z
Step 2: Full SVD on a Small Matrix

**Intuition:**

\[
M = (r + r') \times (r + r') = S \times \Lambda_0 \times Y \times T'
\]

\[
M = L \times \Lambda \times R'
\]

**Details:**

\[
M = S \begin{bmatrix} \Lambda_0 & 0 \\ 0 & Y \end{bmatrix} T' = L \Lambda R'
\]

\[
A = [U_0 \Delta Q] S \begin{bmatrix} \Lambda_0 & 0 \\ 0 & Y \end{bmatrix} T'[V_0 \Delta Z]' = [U_0 \Delta Q] L \Lambda R'T'[V_0 \Delta Z]'
\]

**Property:**

- **Efficiency:** takes \(O((r + r')^3)\) \([r' \ll r]\)
- **Effectiveness:** No extra error
Step 3: Rotate Orthonormal Basis

➡️ Intuition:

Rotate $u_1 u_2 q$ by $L$

Rotate $v_1 v_2 z$ by $R$

➡️ Details:

$U = [U_0 \Delta Q] L$

$V = [V_0 \Delta Z] R$

➡️ Property:

- **Complexity:** $O(nr^2)$
- **Overall SVD Update Complexity:** $O(nr^2 + m'r' + nr'^2)$

Re-compute SVD: $O(nr^2 + mr)$
Analysis and Variants

- Time complexity of \textit{Cheetah-D}: $O(nr^2 + nr'^2 + r^6)$
- \textbf{ARK:} $O(n^2r^4 + mr + r^6)$

Comparison Example \( (n = 3328, r = 500, r' = 5) \)

- ARK: 7.5s
- Ours: 0.4s

Variants

- Undirected graphs
- Attributed graphs

\textit{Cheetah-D} Algorithm Sketch

\begin{verbatim}
t = 1, Initialize SVD of A1 and A2

for t=2,3,…

  Update SVD for A1 \( \leftarrow O(n(r^2 + r'^2)) \)
  Update SVD for A2 \( \leftarrow O(n(r^2 + r'^2)) \)
  Update Ker(A1,A2) \( \leftarrow O(nr^2 + r^6) \)

end
\end{verbatim}
Roadmap

- Motivation
- *Cheetah-D* for Directed Graphs
- Experimental Results
- Conclusion
Case Study — MTA Bus Traffic

Graph construction

- Monitor traffic volume of 30 bus stops on 3 routes, from Monday, 03/24/2014 — Sunday, 03/30/2014
- Represent each stop as a time series where each timestamp is traffic volume within each hour
- On each day, build a graph for the 30 stops using Granger causality test

Graph kernel computation

- Graph kernel is computed between two graphs of two consecutive days
Case Study — MTA Bus Traffic

Accuracy vs. rank:

- The graph kernel is normalized by the number of edges.

Available at http://bustime.mta.info

Accumulated error of our method grows slowly (sublinearly) over time; and (2) the overall accumulated error is very small (less than 0.02%). Notice that, (1) the accumulated error of our method grows slowly (sublinearly) over time; and (2) the overall accumulated error is very small (less than 0.02%).

Weekdays Schedule

- Weekdays have different bus schedules during weekdays.
- The kernel goes up on Sunday because traffic patterns differ between weekdays and weekends.
- The graph kernel is normalized by the number of edges.

Weekends Schedule

- Weekends have different bus schedules during weekends.
- The kernel drops sharply between Friday and Saturday. This reflects the fact that traffic volume within each hour as traffic volume at 30 bus stops.

Different since MTA

- Different since MTA

Figure 1: Case study – real time MTA bus traffic

Figure 2: Relative error of update matrix, $U_r$.

Figure 3: Relative error of update matrix, $U'_r$.

Weekends Schedule

- Weekends have different bus schedules during weekends.
- The kernel drops sharply between Friday and Saturday. This reflects the fact that traffic volume within each hour as traffic volume at 30 bus stops.

Different since MTA

- Different since MTA
Relative Error

All cases, Err <0.02%

\( \hat{n} = 4183, \hat{m} = 5692 \)
Avg Error vs. Rank

When $r>50$, $\text{Err}<0.05\%$

$\bar{n} = 4183, \bar{m} = 5692$
Running Time vs. Rank

- **ARK-U+**
- **OURS**

**Figure 4:** Average error vs. reduced rank $r$. Each curve has different reduced rank $r_0$ for the update matrix in $\text{UpdateEigen}$.

**Figure 5:** Running time of Cheetah-U on AS with different reduced rank $r$. We run the above experiment under different $r$ and average the relative error over 10 time stamps. To see how the approximation of the updates affects the accuracy, we also vary the reduced rank $r_0$. As can be seen from Figure 4, the error quickly drops when $r$ increases.

**5.3 Efficiency Results**

- **Running time vs. rank:** We compare the speed of Cheetah-U with ARK-U+ proposed in [18] varying reduced rank $r$ and average the running time over 10 time stamps. We set reduced rank of update matrix as $r_0 = 5$. Figure 5 clearly shows that our method is much faster than ARK-U+.

- **Scalability:** In order to evaluate the scalability of our method, we run Cheetah-U on graphs with different sizes $n$. Figure 6 shows the running time under different $r$ while fixing $r_0 = 5$. Figure 7 shows the running time under different $r_0$ while fixing $r = 100$. In both figures, we can see that the running time grows linearly wrt the size of the input graphs, which is consistent with our complexity analysis in Theorem 3.2.

- **Quality vs. speed:** Finally, we evaluate how the proposed method balances between the quality and speed. In Figure 8, we show relative error vs. running time of different methods. Each dot in the figure is with different reduced rank $r$. Clearly, our method achieves the best trade-off between quality and time.

**6 Related Work**

- **Graph Kernel.** Graph kernel provides an expressive and non-trivial measure of similarity on graphs (see [4] for a comprehensive review). It has seen applications ranging from automated reasoning [31] to bioinformatics/chemoinformatics [11, 26]. A recent interesting work uses graph kernel to address team member replacement problem [22]. According to what substrstructures used for comparison in two graphs, graph kernels...
Scalability

Figure 4: Average error vs. reduced rank $r$. Each curve has different reduced rank $r$ for the update matrix in UpdateEigen.

Figure 5: Running time of Cheetah-U on AS with different reduced rank $r$. We run the above experiment under different $r$ and average the relative error over 10 time stamps. To see how the approximation of the updates affects the accuracy, we also vary the reduced rank $r$. As can be seen from Figure 4, the error quickly drops when $r$ increases.

5.3 Efficiency Results

Running time vs. rank: We compare the speed of Cheetah-U with ARK-U+ proposed in [18] varying reduced rank $r$ and average the running time over 10 time stamps. We set reduced rank of update matrix as $r_0 = 5$. Figure 5 clearly shows that our method is much faster than ARK-U+.

Scalability: In order to evaluate the scalability of our method, we run Cheetah-U on graphs with different sizes $n$. Figure 6 shows the running time under different $r$ while fixing $r_0 = 5$. Figure 7 shows the running time under different $r_0$ while fixing $r = 100$. In both figures, we can see that the running time grows linearly wrt the size of the input graphs, which is consistent with our complexity analysis in Theorem 3.2.

Quality vs. speed: Finally, we evaluate how the proposed method balances between the quality and speed. In Figure 8, we show relative error vs. running time of different methods. Each dot in the figure is with different reduced rank $r$. Clearly, our method achieves the best trade-off between quality and time.

6 Related Work

In this section, we review the related work in terms of (a) graph kernel, (b) dynamic graph mining.

Graph Kernel. Graph kernel provides an expressive and non-trivial measure of similarity on graphs (see [4] for a comprehensive review). It has seen applications ranging from automated reasoning [31] to bioinformatics/chemoinformatics [11, 26]. A recent interesting work uses graph kernel to address team member replacement problem [22]. According to what substructures used for comparison in two graphs, graph kernels scale near linearly.
Quality vs. Speed

The diagram illustrates the trade-off between relative error and running time for different approaches: ARK-U+, Cheetah-U, First-order, and Second-order.

- Ours: The least relative error with a running time of around 0.1 seconds.
- ARK-U+: A higher relative error with a running time of around 5 seconds.
- Cheetah-U: Intermediate relative error with a running time of around 1.5 seconds.
- First-order: Higher relative error with a running time of around 3 seconds.
- Second-order: Highest relative error with a running time of around 7 seconds.

The graph shows that Ours offers the best balance between accuracy and speed compared to the other methods.

---

**References**

- A family of methods on AS.
- ARK-U+
- Cheetah-U
- First-order
- Second-order

---

**Figure 8** Relative error vs. running time of comparison
Roadmap

- Motivation

- Cheetah-D for Directed Graphs

- Experimental Results

- Conclusion
Conclusion

- **Goal**: track graph kernel of dynamic graphs
- **Our Solution**: *Cheetah-D*
  - **Key idea**: track low-rank approx
  - **Results**:
    - ★ Complexity: $O(nr^2 + nr'^2 + r^6)$
    - ★ In practice: ~15x faster, Err<0.05%
  - **More in paper**:
    - ★ *Cheetah-U* for undirected graphs
    - ★ Error bound analysis