New Approaches for Performance Optimization and Analysis of Large-Scale Dynamic Social Network Analysis using Anytime Anywhere Algorithms

Eunice E. Santos[a], Vairavan Murugappan[a] and John Korah[b]

[a] School of Information Sciences, University of Illinois at Urbana-Champaign
[b] Department of Computer Science, California State Polytechnic University Pomona

Social Network Analysis

- Dramatic increase in the availability of dynamic data from various information sources
  - Example: Social media networks, smart sensors, stock market, etc.

- Facebook (over 2.3 billion active users[a]), Twitter (300 million active users[b]), LinkedIn (610 million users[c])
  - Network dynamism - relationships, followers and connections
  - Continuously evolving networks

- Opportunities and Applications:
  - Large datasets significantly extends our understanding of underlying social phenomenon
  - Disaster management, Health care, Business analytics

[a] http://newsroom.fb.com/company-info/ [b] https://about.twitter.com/company [c] https://press.linkedin.com/about-linkedin

Challenges

Network size:

- Computation time and resources increases dramatically with network size
- Restricts the utility of social network analysis (SNA) in time critical applications

Network Dynamism:

- On average about 500 million new tweets every day[a]
- In many real-time social media analytics and disaster management, the underlying network is evolving
- Restarting or analyzing static snapshot of the network will often yield poor performance

Load Balancing:

- Distributed storage and dynamism causes load imbalance
- Most social networks are small-world networks and exhibit power law characteristics

[a] http://www.internetlivestats.com/twitter-statistics/

Our Focus

- Handle dynamic changes as quickly as possible
- Maximize the accuracy of the network state and analysis
- Reduce overhead during load balancing
- Key idea is to balance the workload and reduce idleness without network repartitioning

Current Works

- Dynamic graph partitioning methods
  - Involves some form of data migration to reduce imbalance
  - Vertex migration, Label propagation, Repartitioning
  - [Khayyat2013, Tsourakakis2014, Khandelwal2017]

- Mizan [Khayyat2013], a graph load balancing system migrates the vertices to different processors based on the runtime metrics such as the number of outgoing messages, incoming messages and response time for each step.

- Vaquero *et al.* [Vaquero2013] proposed a load balancing system where the vertex migration is decided based on the number of neighbors.

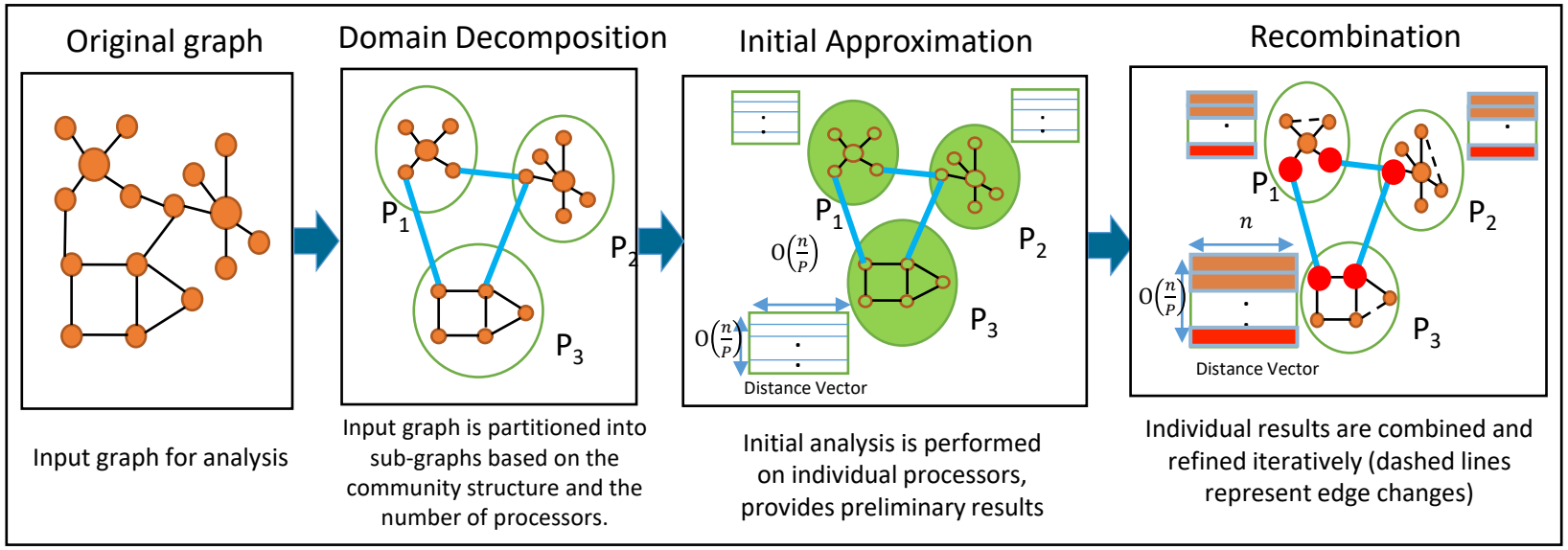
- Hermes [Nicoara2015] , provides a dynamic graph repartitioning algorithm to reduce the number of edges between partitions. However, their main focus is on providing graph management rather than graph analysis.

Anytime Anywhere Framework for Network Analysis

- Designing efficient parallel/distributed algorithms for
  - Handling large and dynamic network analysis.
  - Efficiently incorporate dynamic changes and minimize recomputations.
  - Providing non-trivial intermediate results.
  - Computational platform independent.

- Centrality, is a key measure to understand and analyze actor roles in social network.
  - Used to identify influential and critical actors in the network.
  - Various centrality measures: Degree centrality, Closeness centrality [Yannick2009], Betweenness centrality

Anytime Anywhere Phases for Network Analysis

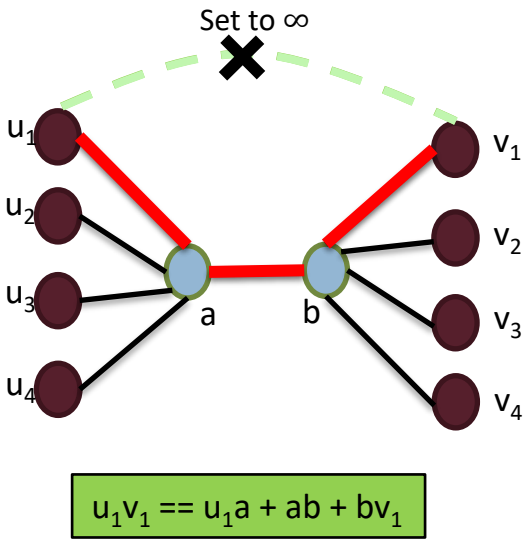


Anytime Anywhere Architecture [Santos et al. 2006, 2006a, 2016, 2016a, 2017a, 2018]

Edge deletion algorithm - Recap

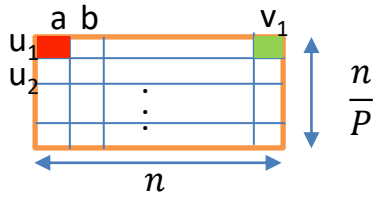
- In this work we focus on edge deletion [Santos2016]
  - Has one of the higher workloads but does not create a large memory imbalance among processors
- Algorithm
  1. Communicate edge to be deleted along with the target node's distance vector to all processor
  2. Identify affected paths in all processors and reset it (to ∞)
  3. Recalculate all the affected paths using the neighbors distance vector.

Edge Deletion



$u_1v_1 == u_1a + ab + bv_1$

Distance Vector in processor of u_1



Distance Vector of vertex b

Edge Deletion – Pseudo-code

Recalculate affected shortest paths

WHILE Q is not empty

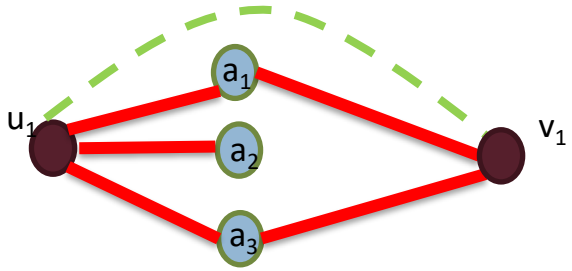
Dequeue (u, v) from Q

FOR EACH neighbor u' of u in sub-graph G_i

IF $DV_i[u][v] > DV_i[u][u'] + DV_i[u'][v]$

$DV_i[u][v] = DV_i[u][u'] + DV_i[u'][v]$

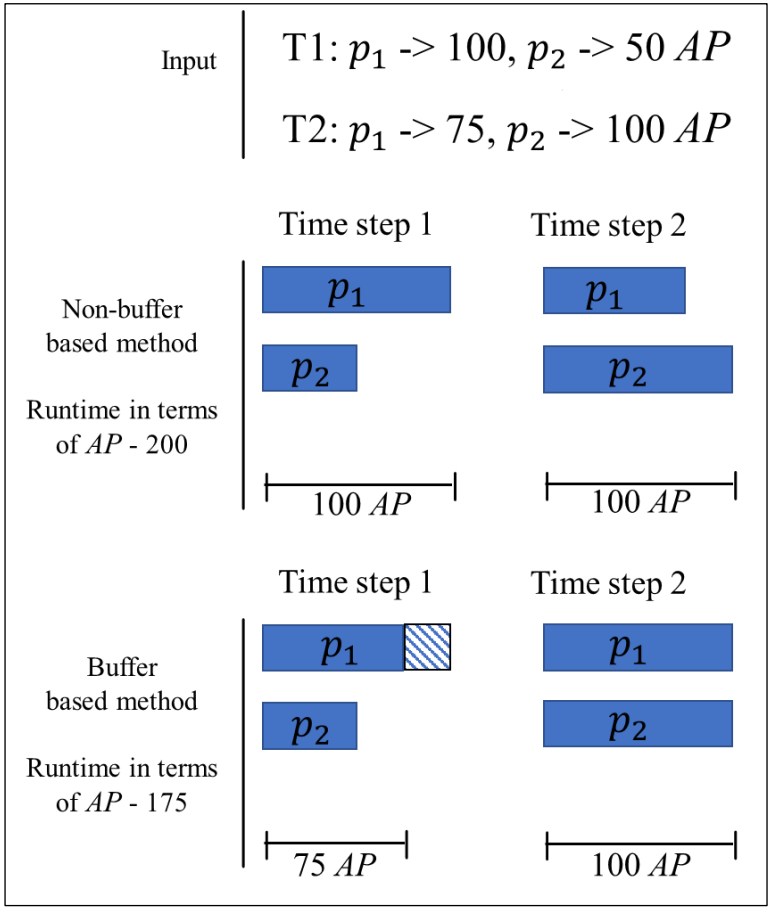
mark $DV_i[u][v]$as updated



u_1v_1 == min(a_1v_1, a_2v_1, a_3v_1)

Deferring Changes

- Balance the number of affected paths recalculated across processors in each iteration.
  - Portion of the workload is moved to future iterations
  - Reduces load imbalance and idleness among processors
- Figure shows handling average number of affected paths *(AP)*
  - *Non-buffer-based method*: Recalculate all the affected path in each iteration
  - *Buffer-based method*: Balances the number of affected paths recalculated
- Constraints
  - Max Buffer Size (B), in terms of number of the affected paths
  - Max number of recombination steps that an affected path can be deferred (T)



Deferring Changes

$$|H'_{i,k}| = \begin{cases} \min\left(\max\left(\left\lceil \frac{\sum_{j=1toP}|\widehat{H}_{j,k}|}{P} \right\rceil, |\widehat{H}_{i,k}| - B\right), |\widehat{H}_{i,k}|\right), k \leq T \\ \min\left(\max\left(\left\lceil \frac{\sum_{j=1toP}|\widehat{H}_{j,k}|}{P} \right\rceil, |\widehat{H}_{i,k}| - B, |\widehat{H}_{i,k-T}| - \sum_{r=k-T}^{k-1}|H'_{i,r}|\right), |\widehat{H}_{i,k}|\right), k > T \end{cases}$$

- Where,
  - $|H'_{i,k}|$ is the number of affected paths selected to be recalculated on processor p_i at iteration k
  - $|\widehat{H}_{i,k}|$ is the number of overall affected paths on processor p_i at iteration k that are available to be recalculated, including the ones carried over from previous iterations.
  - P is the number of processors
  - B and T are the constraints

Deferring Changes

$$|H'_{i,k}| = \begin{cases} \min\left(\max\left(\left\lceil\frac{\sum_{j=1toP}|\hat{H}_{j,k}|}{P}\right\rceil, |\hat{H}_{i,k}| - B\right), |\hat{H}_{i,k}|\right), k \leq T \\ \min\left(\max\left(\left\lceil\frac{\sum_{j=1toP}|\hat{H}_{j,k}|}{P}\right\rceil, |\hat{H}_{i,k}| - B, |\hat{H}_{i,k-T}| - \sum_{r=k-T}^{k-1}|H'_{i,r}|\right), |\hat{H}_{i,k}|\right), k > T \end{cases}$$

- Average number of affected paths across all processors that are available to recalculate

Deferring Changes

$$|H'_{i,k}| = \begin{cases} \min\left(\max\left(\left\lceil\frac{\sum_{j=1toP}|\widehat{H}_{j,k}|}{P}\right\rceil, |\widehat{H}_{i,k}| - B\right), |\widehat{H}_{i,k}|\right), k \le T \\ \min\left(\max\left(\left\lceil\frac{\sum_{j=1toP}|\widehat{H}_{j,k}|}{P}\right\rceil, |\widehat{H}_{i,k}| - B, |\widehat{H}_{i,k-T}| - \sum_{r=k-T}^{k-1}|H'_{i,r}|\right), |\widehat{H}_{i,k}|\right), k > T \end{cases}$$

- Average number of affected paths across all processors that are available to recalculate
- To maintain the buffer constraint *B*

Deferring Changes

$$|H'_{i,k}| = \begin{cases} \min\left(\max\left(\left\lceil \frac{\sum_{j=1 to P} |\widehat{H}_{j,k}|}{P} \right\rceil, |\widehat{H}_{i,k}| - B\right), |\widehat{H}_{i,k}|\right), k \leq T \\ \min\left(\max\left(\left\lceil \frac{\sum_{j=1 to P} |\widehat{H}_{j,k}|}{P} \right\rceil, |\widehat{H}_{i,k}| - B, |\widehat{H}_{i,k-T}| - \sum_{r=k-T}^{k-1} |H'_{i,r}|\right), |\widehat{H}_{i,k}|\right), k > T \end{cases}$$

- Average number of affected paths across all processors that are available to recalculate
- To maintain the buffer constraint *B*
- To ensure that an affected path is not deferred for more than *T* iterations

Buffer-based method

- Two types of Buffer-based method for deferring changes
  - Type A and Type B
  - In both cases the workload for each processor is same for communication, identification and refinement

- *Non-Buffer-based method* – Recalculate all the affected paths using the neighbors distance vector.
- *Type A* – Recalculate affected paths such that the workload is balanced across processors and defer the rest to future iterations
- *Type B* – Recalculate the affected paths only on the processor with the edge

- By performing theoretical analysis we show that
  - For most conditions with reasonable assumptions
  - Our method performs asymptotically no worse than the non-buffer-based method for edge deletion during closeness centrality computation.

Conclusion & Future Directions

- Current load balancing methods focus on vertex migration and dynamic graph partitioning.
- We show that load balancing can also be performed by deferring the changes across time steps.
  - Without incurring the data migration overhead
- In future, we will validate our method experimentally using real-world and synthetic networks
- We will also examine the performance of this approach for other types of changes
  - Vertex additions/deletions, edge additions and edge weight changes

References

1. [Khayyat2013] Z. Khayyat, K. Awara, A. Alonazi, and D. Williams, "Mizan : A System for Dynamic Load Balancing in Large-scale Graph Processing," EuroSys, pp. 169–182, 2013
2. [Tsourakakis2014] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "FENNEL: Streaming graph partitioning for massive scale graphs," in WSDM 2014 - Proceedings of the 7th ACM International Conference on Web Search and Data Mining, 2014
3. [Vaquero2013] L. M. Vaquero and C. Martella, "Adaptive Partitioning of Large-Scale Dynamic Graphs," in Proceedings of the 4th Annual Symposium on Cloud Computing, 2013, p. 35:1-–35:2.
4. [Nicoara2015] D. Nicoara, S. Kamali, K. Daudjee, and L. Chen, "Hermes: Dynamic Partitioning for Distributed Social Network Graph Databases," in EDBT, 2015.
5. [Khandelwal2017] A. Khandelwal, Z. Yang, E. Ye, R. Agarwal, and I. Stoica, "ZipG: A Memory-Efficient Graph Store for Interactive Queries," in Proceedings of the 2017 ACM International Conference on Management of Data, 2017, pp. 1149–1164.
6. [Santos2006] E. E. Santos, L. Pan, D. Arendt, H. Xia, and M. Pittkin, "An Anytime Anywhere Approach for Computing All Pairs Shortest Paths for Social Network Analysis," in Integrated Design and Process Technology, 2006
7. [Santos2006a] E. E. Santos, L. Pan, D. Arendt, and M. Pittkin, "An Effective Anytime Anywhere Parallel Approach for Centrality Measurements in Social Network Analysis," in 2006 IEEE International Conference on Systems, Man and Cybernetics, 2006, vol. 6, pp. 4693–4698.
8. [Santos2008a] L. Pan and E. E. Santos, "An anytime-anywhere approach for maximal clique enumeration in social network analysis," in IEEE International Conference on Systems, Man and Cybernetics , 2008. SMC 2008, 2008, pp. 3529–3535.
9. [Santos2016] E. E. Santos, J. Korah, V. Murugappan, and S. Subramanian, "Efficient Anytime Anywhere Algorithms for Closeness Centrality in Large and Dynamic Graphs," in 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2016, pp. 1821–1830.
10. [Santos2016a]E. E. Santos, J. Korah, V. Murugappan, and S. Subramanian, "Effectively Handling New Relationship Formations in Closeness Centrality Analysis of Social Networks Using Anytime Anywhere Methodology," 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom). pp. 354–361, 2016.
11. [Santos2017a] E. E. Santos, J. Korah, V. Murugappan, and S. Subramanian, "Efficient anytime anywhere algorithms for vertex additions in large and dynamic graphs," in Proceedings - 2017 IEEE 31st International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2017, 2017, pp. 1510–1519.
12. [Santos2018a] E. E. Santos, J. Korah, and V. Murugappan, "Handling Vertex Deletions in Memory Scalable Anytime Anywhere Algorithms for Large and Dynamic Social Networks," in Proceedings - 2018 IEEE 32st International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2018, 2018.