A Distributed Algorithm for Force Directed Edge Bundling

Yves Tuyishime * Yu Pan † Hongfeng Yu ‡
University of Nebraska-Lincoln

ABSTRACT

Existing edge bundling algorithms typically require the global information structure of a graph. Therefore, with a simple division of the edges of a graph, it is challenging to conduct edge bundling in a distributed environment and achieve scalable performance. We select a representative edge bundling algorithm, Force-Directed Edge Bundling (FDEB), and parallelize it in a distributed environment. Particularly, to address the difficulties of partitioning and distributions of a large graph among processors, we first create a high dimensional space to represent the data distribution of a graph in FDEB. Second, we map each edge as a data point in this high dimensional space, and then partition and distribute the point cloud among processors. In this way, we can significantly reduce the data communication across processors, and ensure each processor assigned with a similar workload.

Index Terms: Human-centered computing—Visualization—Visualization techniques—Graph drawings

1 INTRODUCTION

To address the issue of visual clutter for node-link diagrams, researchers have developed *edge bundling* algorithms [1, 2] that visually merge similar edges into curved bundles and can effectively reveal high-level edge patterns with reduced visual clutter. However, these algorithms are typically characterized with high computational complexities and slow speeds for large graphs. For example, Holten et al. developed a Force-Directed Edge Bundling (FDEB) algorithm in which edges are considered as flexible springs that can attract each other while their node positions stay the same, and then a force-directed technique is used to calculate the bundling [2]. However, because the algorithm goes through every pair of edges, its complexity is $\mathcal{O}(n^2)$ where n is the number of edges of a graph. To speed up edge bundling algorithms, the existing efforts have mostly focused on GPU acceleration, while the size of a graph that can be handled is constrained by the available memory of a single machine, and thereby the scalability is limited.

A more scalable solution is to carry out edge bundling using several machines in a distributed environment. However, most existing edge bundling algorithms require the global structure of a graph. Therefore, with a simple division of edges in a graph, it is difficult to achieve balanced workloads and lower inter-processor communication among processors. A naive solution would incur extensive data exchange among processors, which results in poor scalability.

In this work, we parallelize FDEB in a distributed environment. Particularly, to partition and distribute a large graph among processors, we first create a high dimensional space to represent the data distribution of a graph in FDEB. Second, we map each edge as a data point in this high dimensional space, and then partition and distribute the point cloud among processors. In this way, we can significantly reduce the data communication across processors, and ensure each processor assigned with a similar workload. Our experimental results demonstrate the scalability of the algorithm.

*e-mail: ytuyishime@cse.unl.edu
†e-mail: ypan@cse.unl.edu
‡e-mail: yu@cse.unl.edu

2 RELATED WORK

Holten first proposes the use of Hierarchical Edge Bundling (HEB) for graphs that contain hierarchy [1]. Later, Holten et al. [2] propose Force Directed Edge Bundling (FDEB) for general graphs. Many edge bundling algorithms have been proposed. However, they are typically characterized with high computational complexities and can take several to hundreds of seconds to generate bundles for large graphs. Only a few attempts have been perceived to address this performance issue. Zhu et al. [4] propose a parallelized FDEB on the GPU (GPUFDEB). Wu et al. [3] use GPU textures to encode a graph and use GPU to carry out force-directed edge bundling. However, these solutions mostly leverage multithreading techniques and require the data of graphs and intermitted calculation results to be held in the GPU memory, which makes it difficult to tackle graphs that are not held in a single machine.

3 DISTRIBUTED EDGE BUNDLING

3.1 High-Dimensional Representation

The original FDEB algorithm needs to scan each pair of edges to determine their compatibility values. We find that it is not necessary to apply this pairwise operation for all edges if we have a distance measure for all edges. Let us consider angle compatibility first[1]. Given an edge l with its start point p and its endpoint q, we can easily gain its normalized vector $v = (q-p)/||q-p||$, which can be mapped to a point on a unit circle. In this way, the angle compatibility between any two edges can be estimated as the arc length between two corresponding points on the unit circle in a 2D space. Figure 1 shows a simple example with five edges. We can see that the angles of l_2, l_5, and l_4 are close to zero, and thereby their corresponding points are close to the x axis of the 2D space. Meanwhile, the points of l_1 and l_3 are close to each other, but are relatively far away for the other three points in the 2D space. Given this 2D representation, we can easily compute the distance between any pair of points. More importantly, it facilitates us to partition the points according to their distribution. For example, Figure 1 clearly shows two distinct groups with respect to their angles.

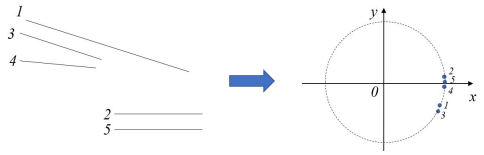


Figure 1: Map the edges into a 2D space according to their angles.

We can extend this idea to create a space for each of the edge compatibility measures. For scale compatibility, we can create a 1D space and map each edge as a point in this 1D space according to its scale or length. Figure 2 shows the mapping of the simple example with five edges. We can see that the scale of l_1 is considerably larger than the other four edges. We can easily partition the points into two groups according to their scales in this 1D space. For position

[1]For simplicity, we consider 2D graphs in this work.

978-1-7281-8468-5/20/$31.00 ©2020 IEEE
DOI 10.1109/LDAV51489.2020.00013

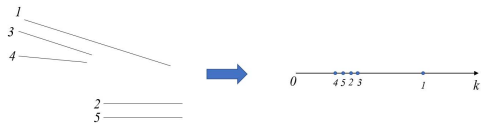


Figure 2: Map the edges into a 1D space according to their scales.

compatibility, we create a 2D space and map each edge as a point in this 2D space according to the positions of its middle point. In this space, we can easily measure the distance with respect to the positions between any two edges.

The total compatibility of two edges is the multiplication of the individual compatibility measure. This inspires us that we can create a high-dimensional space by combining the point distribution in the space of each compatibility measure. We can then map each edge as a data point in this high-dimensional space, and measure their distance as their total compatibility among any pair of edges. Given the space of each compatibility measure, we can obtain a 5D space.

We did not handle visibility compatibility in the current work due to the difficulty in designing its representation. By introducing additional dimensions, we can gain finer partitioning results. The current partitioning can be regarded as a coarser result with a larger bound, and does not compromise our data partitioning.

3.2 Distributed FDEB

Given our high-dimensional representation, we can carry out FDEB in a distributed environment by increasing the performance and avoiding data duplication among the processors.

First, given an input graph, we map each edge into a point in our 5D space where each subspace encodes the data distribution with respect to each individual compatibility measure. The distance between any two points in this high-dimensional space is proportional to the compatibility measure between the two corresponding edges.

Second, we partition the point cloud using the K-D tree that can ensure that each partition has a similar number of points. The number of partitions is equal to the number of processors. For simplicity, we assume that the number of processors is an exact power-of-two. In this case, if we assign the points within a partition to a processor, the processors can have a similar number of edges to be bundled. To avoid data exchange among the processors, we create a ghost area for each partition such that a processor can always find the corresponding interacting edges of its local data without fetching data from any remote processors, thus minimizing the data communication cost.

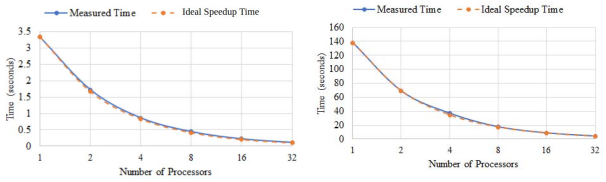
Third, each processor computes edge compatibility measures for its local data. With our high-dimensional partitioning, a processor can find the pairwise interacting edges within its local data.

Fourth, each processor conducts force-directed edge bundling based its local edge compatibility measures. We note that each processor does not bundle the edges within the ghost area, but use these edges to bundle its local edges. The edges within the ghost area are bundled in remote processors. In this way, although the edges in the ghost area may be different, the total number of local bundled edges is approximately equal for each processor. This ensures balanced workloads among processors.

Finally, the partial edge bundling results generated at each processor are then aggregated. This is simply implemented using distributed gather operations (e.g., MPI_Gather).

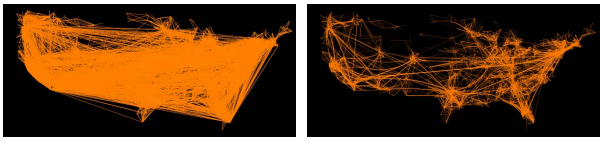
4 Results

We implement our algorithm using MPI and C++ and experiment with it on Crane, a supercomputer operated by the Holland Computing Center at the University of Nebraska-Lincoln. We used a US airline graph (2100 edges) and a US migration graph (9780 edges) to design and conduct our scalability experiment.



(a) (b)

Figure 3: Speedup of our scalability experiment to bundle (a) the US airline graph and (b) the US migration graph. In each plot, the horizontal axis presents the number of processors, and the vertical axis represents the running time in seconds. In each plot, we show the measured time and ideal speedup time.



(a) (b)

Figure 4: Visualization results of the US migration graph using (a) node-link diagram and (b) edge bundling.

Figure 3 shows the performance of distributed FDEB carried out on the processors. Thanks to our high-dimensional representation and partitioning approach, there are no communications required among the processors, and each processor has a balanced workload. Figure 3 conveys that our distributed algorithm achieves almost ideal speedup, and the parallel efficiencies are 94% and 97.6% (32 processors vs. 1 processor) for the US airline graph and the US migration graph, respectively.

Figure 4 shows the visualization results of the US migration graph. It is hard to perceive clear patterns from the node-link diagram in Figure 4(a) due to the visual clutter problem. Figure 4(b) shows the edge bundling result, which significantly reduces the visual clutter and is close to the ones generated by the original FDEB on a single machine. However, our method can generate these results in a more scalable fashion.

5 Conclusion

Our method is simple and easy to understand and implement. We will study visibility compatibility and larger graphs in our next development and experiment to gain a deeper understanding of the impact of data partitioning and distribution schemes on conducting distributed edge bundling. Although we design the approach for FDEB, we expect that the parallelization methodology developed in this work can be extended to other edge bundling algorithms.

Acknowledgments

This research has been sponsored by the National Science Foundation through the grant IIS-1652846.

References

[1] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on visualization and computer graphics*, 12(5):741–748, 2006.

[2] D. Holten and J. J. Van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[3] J. Wu, L. Yu, and H. Yu. Texture-based edge bundling: A web-based approach for interactively visualizing large graphs. In *2015 IEEE International Conference on Big Data (Big Data)*, pp. 2501–2508, 2015.

[4] D. Zhu, K. Wu, D. Guo, and Y. Chen. Parallelized Force-Directed Edge Bundling on the GPU. In *2012 11th International Symposium on Distributed Computing and Applications to Business, Engineering Science*, pp. 52–56, 2012.