BSc project proposals

in

Algorithms, Data Structures and Foundations of Machine Learning

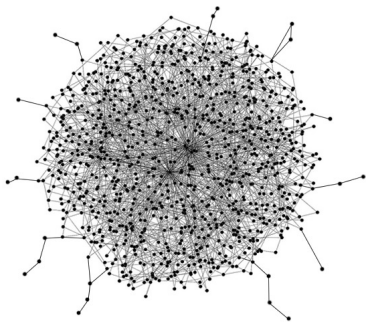
Peyman Afshani
Gerth Stølting Brodal
Kasper Green Larsen
Chris Schwiegelshohn

{peyman, gerth, larsen, schwiegelshohn}@cs.au.dk
Nygaard 3^{rd} floor

The following pages contain potential topics for BSc projects, but other projects are also possible. The final topic and direction of the project is settled under guidance by the advisor. In particular the weightening between theory and practical implementations is done on an individual basis, and is often adjusted during the BSc project process. Please do not hesitate to pass by our offices on or send us an e-mail for setting up a meeting to discuss potential BSc projects.

Connectivity in Graph Sketching

Problem *Let $G(V, E)$ be a graph over n vertices, where n is known. We process edge insertions and deletion of the form ($A_{u,v} + 1$ signifying an insertion and ($A_{u,v} - 1$ signifying a deletion, where A is the adjacency matrix of G.*
Our goal is to determine whether the graph is connected or not, using as little space as possible.



Graph streaming is a model of computation for very large graphs. Here, we process a sequence of edge insertions and deletions. At any point, we want to be able to query certain properties of the graph. The challenge is to do so in little space, i.e. we want to store roughly $O(n)$ bits of space, instead of the maximum size of the graph which corresponds to $O(n^2)$ space.

We focus on the connectivity problem, where we want to decide whether a graph is connected or not. If we are only processing edge insertion, it is very simple to decide whether the graph is connected with union find data structures. Details we leave as an exercise to the interested reader.

If we also process edge deletions, the problem becomes very challenging. Perhaps surprisingly, even in this case, $O(n \log^3 n)$ bits of space are enough!

The aim of the project is to understand, reproduce and implement the classic algorithm as well as a few heuristics, run them on real world data sets and compare their performance.

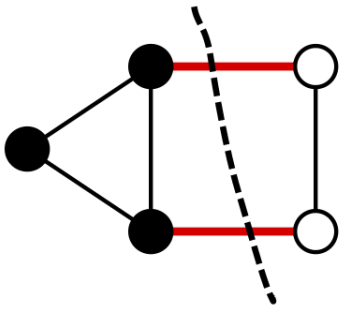
References

- *Analyzing graph structure via linear measurements.* Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012, pages 459–467.
  DOI: 10.1137/1.9781611973099.40.

Contact: Chris Schwiegelshohn

Two-Pass Min-Cut in Graph Streaming

Problem *Let $G(V, E)$ be a graph over n vertices, where n is known. We process a sequence of edge insertions. At the end we want to output a global min cut of the graph, i.e. a partition of the graph into two sets S and $V \setminus S$ such that $|E \cap (S \times V \setminus S)|$ is minimized. Our goal is to use as little space and as few passes over the data as possible, while solving this problem* exactly.



Graph streaming is a model of computation for very large graphs. Here, we process a sequence of edge insertions and deletions. At any point, we want to be able to query certain properties of the graph. The challenge is to do so in little space, i.e. we want to store roughly $O(n)$ bits of space, instead of the maximum size of the graph which corresponds to $O(n^2)$ space.

We focus on the min-cut problem, i.e. a partition of the graph into two sets S and $V \setminus S$ such that $|E \cap (S \times V \setminus S)|$ is minimized. One can show that if we use a single pass over the data, finding the optimal min-cut requires $\Omega(n^2)$ space, i.e. we cannot do substantially better than simply storing everything. Perhaps surprisingly, two passes are enough to solve this problem in no more than $O(n \log^2 n)$ space!

The aim of this project will be to give an exposition and literature review on this problem, both for exact and approximate guarantees and compare the different techniques.

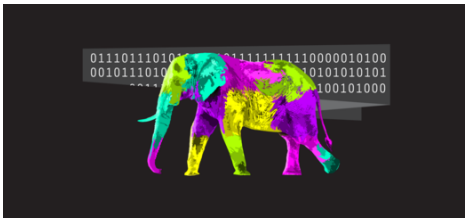
References

- *A Simple Semi-Streaming Algorithm for Global Minimum Cuts*. Sepehr Assadi and Aditi Dudeja, 4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021, pages 172–180.
  DOI: dl.acm.org/doi/10.1137/1.9781611976496.19.

- *Intractability of min- and max-cut in streaming graphs*, Mariano Zelke, Inf. Process. Lett., 111(3), 145–150, 2011.
  DOI: 10.1016/j.ipl.2010.10.017.

Contact: Chris Schwiegelshohn

Tracking Frequent Items in Data Streams



Simons Institute

Finding frequently occurring items in a dataset is a common task in data mining. In the data streaming literature, this problem is typically referred to as the heavy-hitters problem, which is as follows: a huge stream of items is processed from one end to the other. Each item in the stream can be thought of as an integer and the goal is to report the integers occurring most often while using limited memory. An integer in the stream may represent e.g. a source IP address of a TCP packet. One then may want to find sources with high utilization in a network traffic monitoring application. Another example has each integer representing how many times a concrete web surfer clicked a web ad on a particular site. The goal then becomes to identify the frequent ad clickers in web advertising. Common to these applications is that one wants a solution with tiny memory consumption and processing time.

The aim of the project is to work with both real-life and synthetic data streams, and to implement, experiment with and compare different solutions to the heavy-hitters problem. An important aspect of several of the solutions to the heavy-hitters problem is that they rely on randomization. Thus a central task is to use basic probability theory to give a theoretical analysis of the solutions. The final report should include a theoretical section covering the basic analysis of different algorithms for the problem, a summary of the implemented algorithms, an experimental evaluation of the algorithms and a discussion of the obtained results.

References

- Lecture notes by Kasper Green Larsen and from U.C. Berkeley.
- *An Improved Data Stream Summary: The Count-min Sketch and its Applications.* Graham Cormode, S. Muthukrishnan. Journal of Algorithms, 55(1), 58-75, 2005. DOI: 10.1016/j.jalgor.2003.12.001.
- *Finding Repeated Elements.* J. Misra, David Gries. Science of Computer Programming, 2(2), 143-152, 143-152, 1982. DOI: 10.1016/0167-6423(82)90012-0.

Contact: Kasper Green Larsen

Closest Points

Problem *Let P be a set of n points in the $\mathbb{R}^d$, given by their coordinates. The problem is to find two points $p, q \in P$ such that the Euclidean distance between them is the smallest.*

This is a classical computational geometry problem. Clearly, we can find the closest pair by going through all the pairs of points, compute the distance between every pair and then find the minimum among them. Unfortunately, this is a rather slow algorithm which runs in $\Theta(n^2)$ time.

The first elegant solution came by Shamos and later published together with Hoey in 1975. Their algorithm was a simple divide and conquer algorithm: partition the point set into two equal sizes, P_ℓ and P_r, e.g., with a vertical line h and then find the closest pair in each subset recursively. Let d be the smallest distance that we find in this way. If the closest pair is completely inside one subset, we are done. Otherwise, one point from P_ℓ and another point from P_r must be the closest points. In particular, we only need to consider the points within distance d of the line h. Now, they observed that only a $O(n)$ pairs need to be considered as each point in P_ℓ or P_r cannot have too many points of distance d next to them. This gives the recursion

$$f(n) = 2f(n/2) + O(n)$$

which solves to $f(n) = O(n \log n)$.

This solution generalizes to a point set in 3D in a more or less straightforward way and it gives an algorithm with $O(n \log^2 n)$ running time. The running time comes from a recursion

$$g(n) = 2g(n/2) + f(n)$$

where $f(n)$ is the time it takes to solve a 2D closest pair problem. If we go with $f(n) = O(n \log n)$, then we get $g(n) = O(n \log^2 n)$. However, a brilliant solution by Shamos and Bentley showed that this can actually be improved to $O(n \log n)$.

The goal of this project is to follow these developments. The tasks are: (1) Implement the $O(n \log n)$ algorithm for finding the closest pair in a set of n points in the plane. (2) Implement an $O(n \log^2 n)$ algorithm for finding the closest pair in a set of n points in 3D. (3) Implement the algorithm by Bentley and Shamos that runs in $O(n \log n)$ time to find the closest pair in a set of n points in 3D.

References

- *Divide-and-Conquer in Multidimensional Space.* Jon Louis Bentley and Michael Ian Shamos. 8th Annual ACM Symposium on Theory of Computing (STOC), 220-230, 1976. DOI: 10.1145/800113.803652.
- *Closest-Point Problems.* Michael Ian Shamos and Dan Hoey. 16th Annual Symposium on Foundations of Computer Science (FOCS), 151-162, 1975. DOI: 10.1109/SFCS.1975.8.

Contact: Peyman Afshani

Smallest Enclosing Ball

Problem *Let P be a set of n points in d-dimensions, given by their coordinates. The problem is to find the smallest ball that contains all the points. Thus, the output is the center of the ball and its radius.*

This is one of the classical problems in clustering. If we assume d is a constant, then the problem can be solved in linear time but often the dependencies are exponential in d. In particular, the best algorithm has the running time of $O(d^2n + 2^{O(\sqrt{\log n})})$ which is exponential in d. As a result, when the dimension is large, settling for approximations is a reasonable choice.

In this project, you will implement and compare two simple solutions.

1. This algorithm achieves a $(1+\varepsilon)$ factor approximation in time $O(nd/\varepsilon + 1/\varepsilon^5)$ (Bâdoiu and Clarkson, 2003).

2. However, there is also a simpler algorithm that achieves a 3/2 factor approximation in linear time, in fact, using only one scan of the input (Zarrabi-Zadeh and Chan, 2006).

The goal of this project is to implement these solutions and to compare them.

Task one. Implement the algorithms. Run some basic tests to make sure that they are correct.

Task two. Run some experiments to measure the running time, and the quality of the solution. Note that one algorithm should be fast but less accurate while the other one should be the opposite.

Task three. In your report, explain briefly how you implemented the algorithms and list any difficulties in doing so. You also need to briefly explain how the algorithms work but you don't have to cover the proof of correctness parts. Were the descriptions of the algorithms sufficient for the implementation or did you have to make some non-obvious choices? Describe how you designed your test cases and how you generated the input. Then, try to justify the results of the experiments.

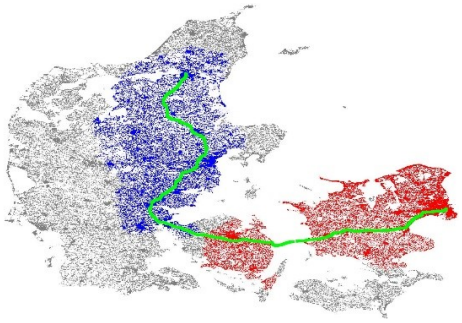
References

- *Smaller Core-sets for Balls.* Mihai Bâdoiu and Kenneth L. Clarkson. Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 801–802, 2003. https://dl.acm.org/citation.cfm?id=644240

- *A Simple Streaming Algorithm for Minimum Enclosing Balls.* Hamid Zarrabi-Zadeh and Timothy Chan. Proc. 18th Canadian Conference on Computational Geometry (CCCG), August, 2006.
http://www.cs.queensu.ca/cccg/papers/cccg36.pdf

Contact: Peyman Afshani

Shortest Paths Computations on Open Street Map Data



Martin Jacobsen

Open Street Map data contains a very detailed description of road networks worldwide and is the underlying data use in many map applications. The data is annotated with both geometric features and logically information of e.g. connection of road segments, speed limits, road types, one-way streets e.t.c.

The aim of this project is to be able to work with open street map data, in particular to visualize open street map data, convert open street map data to graph representations, implement algorithms for finding shortest paths in road network graphs, and to evaluate experimentally the performance of various algorithms. An important part of the process will be to read the theory behind selected shortest path algorithm, e.g. Dijkstra's algorithm, bidirectional shortest path algorithms, A*, landmark based algorithms, contraction hierarchies, highway hierarchies, hub labelling, and transit node algorithms. The final report should include a summary of the implemented theory, a description of the implementation, an experimental evaluation of the implemented algorithms, and a discussion of the obtained results — in particular a discussion of space usage versus query time.

The Open Street Map data can also be combined with other data sources, like the Danish height elevation model, to take elevation differences into account for e.g. finding optimal bicycle routes.

References

- *Route Planning in Transportation Networks.* Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, Renato F. Werneck. Algorithm Engineering 2016: 19-80. DOI: 10.1007/978-3-319-49487-6_2.
- Videos with Andrew Goldberg on the topic: Beyond Worst Case Analysis Workshop 2011, NWU 2013.
- Slides from the Algorithm Engineering course, 2017: route.pdf.

Contact: Gerth Stølting Brodal.