CS 598 JGE ✧ Fall 2017

One-Dimensional Computational Topology

Project Proposals

CS 598JGE Project Proposal

Name and Netid: Bhuvan Venkatesh: bvenkat2

Introduction

Embedding arbitrary graphs into Hypercubes has been the study of research for many practical implementation purposes such as interprocess communication or resource. A hypercube graph is any graph that has a set of 2^d vertexes and all the vertexes are labeled $0, 1, ..2^d$ in binary; an edge is between two vertexes if their binary representations differ by a digit. When we say embedding given a graph G, we are asking which dimension hypercube graph contains G as a subgraph. The problem is not the isometric graph embedding, an embedding that preserves all pairwise distances – this version will be defined later. Hypercube graphs are also naturally recursive structures, meaning we could do split up hypercubes recursively. The formal problem statement in terms of the decision problem, optimization problem, and the approximation problem are all in the next section. The last thing to note about hypercube embedding is that it is not truly a graph packing problem because we are not partitioning edges into disjoint subsets – that may be a the next step after solving the embedding problem but does not precede it.

Known Results

Some definitions first. $G = (V, E)$ has an **isometric embedding** of a hypercube $H_n = (A, B)$ [all edge weights 1] if there exists a one-to-one mapping $f : V \to A$ and for all shortest paths between two vertexes uv $d(u, v)$, the embedding preserves the pairwise distances. **We are not looking for isometric**, instead looking for the minimum n such that $V \subseteq A$ and $E \subseteq B$. The approximation problem is given OPT as the minimal dimensional hypercube that embeds graph G, can we find an $f(|V|)$ approximation algorithm, such that the optimum given by our algorithm OPT_f follows $OPT_f/OPT \leq f(|V|)$.

There are some necessary conditions (though not sufficient) that can be recognized in linear time. One of the most paramount is that there are no odd length cycles. Hypercube graphs only contain even length cycles and any odd length cycle cannot be formed from the subgraph. The other condition is the the Hypercube degree must at least be the at least the maximum degree of all vertexes in the graph, but these conditions alone cannot help one either in the decision problem or the optimization problem. The decision problem, does a hypercube graph of dimension n contain a subgraph G, is NP-Complete. It has been shown that the optimization version of the problem is NP-Hard [1]. The reduction is from 3 Exact Set cover, so as far as the structure of the problem there could be some links to Sudoku/N-Queens that can come into play.

In the world of approximation, there are already a few orthogonal problems that have been solved. Given a graph and a set of points in the plane, there is an $O(log(|V|))$ approximation factor algorithm [3] where the one is trying the minimize the length of the edges drawn. There could be some nugget of the internals of the algorithm that can be used in an approximation algorithm with this problem. This function may be easily invertible in the planar case, hopefully holding the same bound.

Special graphs, namely complete trees, balanced trees, and arbitrary trees have their own solved class of problems [5]. Though, some of these are solved for the isometric version of the problem or with a constant factor of dilation d (each edge is allowed to be expanded to a path of at most d edges). Since trees are naturally planar graphs, I feel that some of the techniques that we learned in class could possibly help us. Even more basic, trees are naturally recursive data structures where hypercubes are as well, meaning a very simple greedy approximation algorithm could be used. While noting special graphs, there is a way of embedding a complete graph isometrically into a hypercube [7], and I again imagine a simple greedy recursive hypercube argument could solve this problem for the special case although these complete graphs for $|V| \geq 5$ are not planar.

In the world of isometric graph embedding, David Eppstein published [4] where one can recognize a partial cube in $O(n^2)$ time. I don't believe there has been a link between isometric graph embedding in terms of problem structure, but there could be a bounding function such that the dimension of the hypercube that is reported by Eppstein's algorithm is always $f(|V|)$ away from the true optimum.

This problem may look like it is linked to boxicity as well. boxicity given a graph $G = (V, E)$ is the minimum dimension k such that V has a one to one mapping of axis parallel boxes in $\mathbb{R}^k$ and there is an edge whenever the boxes intersect. There is a sublinear factor approximation algorithm [6] for

solving this problem, but I don't believe that this may not have as much of an equivalence to hypercube problems because the vertexes of a hypercube graph always have the same degree where the boxicity problem need not have that.

Half Baked Ideas

I'm not going to aim for this to be a survey, but these ideas are about a quarter baked, so this may turn into a survey of results

There are approximation algorithms for finding the answer within $O(log(n))$ [but this is for a generalized set of P points] but is there a constant factor approximation algorithm for the hypercube case? It is also unknown whether or not that a constant factor approximation algorithm will lead to a monumental result (P=NP) because of the link to constant factor approximation algorithms for Traveling Salesman implying P=NP. [I will not attempt to solve P vs NP for the final project, though that would be a pretty good final project].

After talking with Jeff, there could be a few different avenues to explore here.

1. If we want to stay with the theme of the class, we could take a look at planar graphs. We could do something with planar separators, recursively find the dimensionality of the corresponding subgraphs and join them back in some higher hypercube construction. At first glance, this may be exponential or impossible given an arbitrary separator. This may be more possible as an approximation algorithm if we make some greedy decisions at each node.

2. The reduction is from 3 exact set cover, a greedy algorithm could obtain a $O(log(n))$-factor approximation for generalized (non-planar) graphs that runs in polynomial time.

3. We could potentially find a constant factor approximation algorithm. This is a very very far fetched idea, but it is based on the recently solved asymetric Traveling Salesman problem which does have a constant factor approximation algorithm [2]. I was think that some kind of reduction – using ATSP as a black box – could get a constant factor approximation algorithm as well.

Plan of Attack

So the plan of attack would be to start out with looking at solved problems form 3 exact set cover and sudoku. If there are approximation algorithms for them, we could use the same arguments to develop one for the hypercube graph. Another avenue in this field could be exploring the relationship of this problem to other NP-Hard optimization problems and seeing if there is a simpler algorithm and argument to make; there still is a lot to learn about the place that this problem holds among all of the other NP-Hard problems.

In terms of the planar cases, we would mostly need to make an original algorithm about sub dividing the graph along a separator, recrusively solving the two parts and joining the graphs in some reasonably fast way. This could turn out to not just to be an approximation algorithm, but an exact embedding given that the graph is planar. If we are to go to the exact embedding, then we may have to do something Tarjan-esque with data structures to both avoid exponential time in the recursion and prove that it is both the lower and upper bound.

In terms of the last bullet point, I think this is only feasible if we have a lot of time and commitment because skimming over the approximation argument for the Asymmetric Traveling Salesman problem, it is not for the feint of heart. Even though we could use this as a black box, I feel like we would need to understand the internals of this to some degree in order to develop a solid algorithm for the hypercube case.

Unofficial Bibliography

Overleaf does not play very nicely with bibtex, sorry for the urliography

[1] https://www.sciencedirect.com/science/article/pii/0166218X9390170S

[2] https://arxiv.org/pdf/1708.04215.pdf

[3] http://ieeexplore.ieee.org/document/63542/

[4] https://arxiv.org/pdf/0705.1025.pdf

[5] $https://ac.els-cdn.com/S0166218X00002560/1-s2.0-S0166218X00002560-main.pdf8$

[6] https://arxiv.org/pdf/1505.04918.pdf
[7] https://www.isima.fr/ beaudou/docs/eurocomb07.pdf

One-Dimensional Computational Photography
Project Proposal

Brendan Wilson
bswilso2

November 10, 2017

Topic

My proposal is to examine the $O(n \log n)$ max-s,t-flow algorithm in directed planar graphs first proposed by Borradaile and Klein [1], using the formulation proposed by Erickson [2]. This algorithm relies on a parameterized s,t-flow to selectively augment a shortest-path tree in the dual graph. At a certain critical value of the parameter, the tree will become disconnected, creating a cycle in the dual. This cycle is a min cut in the primal graph.

This algorithm works by starting with a feasible flow and selectively increasing the parameter until the flow is optimal. In the linear programming sense, this is equivalent to finding a feasible point inside the constraining polyhedron and improving it until it's optimal. There is, of course, the dual way of performing this action. We can instead find an optimal point, and improve our solution until it's within the feasible region.

In the sense of this algorithm, this alternative approach is equivalent to setting the parameter to its maximum possible value. In the general case the flow induced by this will be infeasible, but we could then decrease the parameter until the flow is feasible. It seems reasonable that since there is an $O(n \log n)$ solution to the primal version of this, there may be a similarly efficient solution to this dual formulation. I propose investigating this, with the primary goal of matching the running time of the primal algorithm.

Jeff showed [2] that there is an $\Omega(n^2)$ bound on graphs embedded on the torus using the primal algorithm, so I'd also like to see if a dual version matches this same constraint. A home run for this would be a way to improve on this, but even matching it would be interesting.

Existing Results

As mentioned, the primary inspiration comes from Jeff's work on improving the analysis of Borradaile and Klein [1, 2]. Jeff also extended this work to cover graphs embedded on surfaces of higher genus.

Borradaile's thesis (`http://cs.brown.edu/research/pubs/theses/phd/2008/glencora.pdf`) provides a good reference for the history of planar max flow algorithms, as well as a thorough treatment of the leftmost augmenting path algorithm.

Plan of Attack

Investigating the behavior of the shortest-path tree over all values of λ would need to come first. According to Borradaile by way of Jeff, each edge in the graph pivots into the spanning tree at most 3 times [2]. It's unclear to me what happens to the tree once a critical value of λ is reached, that is, the point where the spanning tree is no longer connected. At this point there is a negative cycle in the parameterized dual graph, so shortest paths are no longer well defined. I'm curious what happens to the remnants of the tree if we keep pushing λ further anyway. One issue here is that the tree is defined in terms of slack, which itself is defined in terms of distance. Slack ceases to be well defined at this point, so an alternative definition will probably be needed, if not an entirely different approach.

Half-baked thoughts

Using the notation from Jeff's paper [2], it seems like as λ is increased the spanning tree "wraps" the s face of the dual, until a cycle is created. This first cycle is the min-cut, but if we continue increasing λ, what happens to this cycle? Does it ever get augmented again? The cycle would be negative in the dual residual network, so we could of course set λ to some arbitrarily high amount and then do negative cycle canceling, but this isn't even close to a good time bound. However if the cycle is never augmented away, then we could maintain the data structure of the shortest-path tree through increasing values of λ and then find all cycles in this structure. Since each vertex in this subgraph necessarily has an in-degree of at most 1, my intuition is that this wouldn't take nearly as long as for a general graph.

Another notion is that the Borradaile-Klein algorithm is essentially just Ford-Fulkerson applied in a strategic way. It may be the case that a dual version of Borradaile-Klein could be considered a specialization of Goldberg and Tarjan's push-relabel algorithm. Who knows?

References

[1] Borradaile, G., and Klein, P. An o(n log n) algorithm for maximum st-flow in a directed planar graph. *J. ACM 56*, 2 (Apr. 2009), 9:1–9:30.

[2] Erickson, J. Maximum flows and parametric shortest paths in planar graphs. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2010), SODA '10, Society for Industrial and Applied Mathematics, pp. 794–804.

Vertex-disjoint Paths with Many Sources and Sinks

Charles Shang (cshang4)

November 11, 2017

1 Introduction

Given a graph G, a set of source vertices S and a set of target vertices T, finding the number of vertex-disjoint paths with one endpoint in S and the other in T is normally solved in general graphs using maximum flow with some modifications [5]. When restricting the problem to planar graphs in order to improve running times, it becomes more complex.

2 Vertex-disjoint paths and Maximum flow

2.1 Vertex-disjoint paths in general graphs

Solving this problem in general graphs is done with two modifications.

The first is adding unit vertex capacities to all non-source non-target vertices. Adding vertex capacities in general graphs was solved very early on by Ford and Fulkerson [3] by simple reduction to remove vertex capacities. This reduction was to replace every vertex v with capacity c with a pair of vertices v_{in}, v_{out} and an edge (v_{in}, v_{out}) with capacity c. Then replace all edges (u, v) with (u, v_{in}) and (v, w) with (v_{out}, w).

The second modification is to add two vertices v_s, v_t with an edge (v_s, s) for each source s and (t, v_t) for each target t.

Both modification only increase the number of vertices and edges by a constant factor and thus do not affect the $O(VE)$ [1] running time of solving maximum flow.

2.2 Vertex-disjoint paths in planar graphs

Solving maximum flow in planar graphs has been done in $O(n \log\log n)$ time [2], which is faster than in the general case. However, attempting to add the same modification to a planar graph to solve the vertex-disjoint path problem can break planarity.

Consider the first modification above, where vertex capacities were added. That reduction no longer works with planar graphs as the resulting graphs may not retain it's planar properties. For example, consider a complete graph with 4 vertices. Performing the above

replacement on any one vertex will result in the underlying undirected graph being complete with 5 vertices. Such a graph can not be planar by Karatowski's Theorem.

Similarly, consider the second modification above. Adding new vertices v_s, v_t can also break planarity. Consider the subgraph containing only vertices in S. If that subgraph consists of 4 fully connected vertices, adding v_s will create a subgraph of 5 fully connected vertices, which again can not be planar by Karatowski's Theorum.

Individually, these problems have been solved. Finding vertex-disjoint paths given a single source and target on surfaces of genus = 0 can be done in $O(n)$ time [6]. More generally, finding the maximum flow given vertex capacities can also be done in $O(n \log n)$ time [4]. Performing maximum flow with multiple sources and targets can be done in $O(n \log^3 n)$ time [7].

3 Proposed work

As far as I know, the existence of a $O(n \text{ polylog } n)$ time solution to finding the maximum number of vertex-disjoint paths in a graph G with an arbitrary number of sources, targets and arbitrary genus is still an open problem [8]. Looking at the results of [7] and [4], it seems like there should be a solution combining their algorithms that would have a running time of $O(n \log^3 n)$. My proposal is to do so.

References

[1] James B. Orlin. 2013. Max flows in O(nm) time, or better. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing* (STOC '13). ACM, New York, NY, USA, 765-774. DOI: https://doi.org/10.1145/2488608.2488705

[2] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. 2011. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the forty-third annual ACM symposium on Theory of computing* (STOC '11). ACM, New York, NY, USA, 313-322. DOI: https://doi.org/10.1145/1993636.1993679

[3] L. R. Ford and D. R. Fulkerson. 1962. *Flows in Networks*. Princeton University Press, New Jersey

[4] H. Kaplan and Y. Nussbaum. 2009. Maximum flow in directed planar graphs with vertex capacities. In *Proceedings of the 17th European Symposium on Algorithms*, pages 397-407

[5] J. Erickson (2014). *Algorithms, Etc.* [online] Jeffe.cs.illinois.edu. Available at: http://jeffe.cs.illinois.edu/teaching/algorithms/ [Accessed 11 Nov. 2017].

[6] H. Ripphausen-Lipa, Dorothea Wagner, and Karsten Weihe. 1993. "The vertex-disjoint menger problem in planar graphs." In *SODA*, vol. 93, pp. 112-119.

[7] G. Borradaile, P. N. Klein, S. Mozes, Y. Nussbaum and C. Wulff-Nilsen, "Multiple-Source Multiple-Sink Maximum Flow in Directed Planar Graphs in Near-Linear Time," In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, Palm Springs, CA, 2011, pp. 170-179. doi: 10.1109/FOCS.2011.73

[8] J. Erickson, P. N. Klein, D. Marx, and C. Mathieu. Algorithms for optimization problems in planar graphs (Dagstuhl seminar 16221). *Dagstuhl Reports*, 6(5):94116, 2016.

CS598JGE: One-Dimensional Computational Topology, Fall 2017

Name: Hsien-Chih Chang (hchang17)

Final Project Proposal

Bichromatic triangle in pseudoline arrangements

A ***pseudoline arrangement*** is a finite collection of simple infinite paths $\ell_i : \mathbb{R} \to \mathbb{R}^2$ (called ***pseudolines***) in the plane, such that all the curves are generic, all the intersections are transverse, and each pair of pseudolines intersects *exactly* once. Without loss of generality all the intersection points lie inside a disk of fixed radius, which after transformation one can assume to be the unit disk. A ***triangle*** in a pseudoline arrangement is a face of size 3.

The following problem was raised by Stefan Felsner during the open problem session in EuroCG 2017 [2]:

> Let $\mathcal{A}$ be a pseudoline arrangement where each pseudoline is colored by either *blue* or *red*; assume both colors are used. Prove that there must be a triangle in $\mathcal{A}$ such that the three bounding paths have both colors.

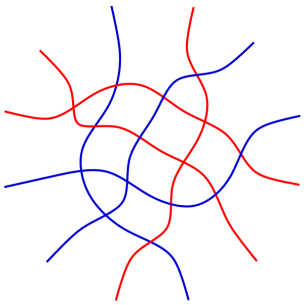

Figure 1. A bichromatic pseudoline arrangement with three bicolored triangles.

Complete history of the problem. No, really. It is an easy exercise to prove that the statement holds for *line arrangements*—collections of Euclidean straight lines—using argument similar to Kelly's proof to the Sylvester-Gallai theorem [1]. Felsner and Pilz [4] provided a proof to the above problem for a subclass of arrangements called *approaching pseudo-lines*. Apart from these I am not aware of any other previous work. Felsner and Pilz [4] mentioned that "... the question for pseudo-line arrangements is by now open for several years."

Motivation. The motivation is threefold. First, my secret goal to this project is actually to get insight to the core lemma in the analysis of Feo and Provan's algorithm [5] for reducing two-terminal planar graphs using electrical transformations. Although there are some key differences between the two problems (one requires each pair of pseudolines to intersect, and the other can only work for a generic closed curve instead of a tangle), the similarity between the two gives me the hope to levitate the intuition from one to the other.

More broadly, the project would give me a chance to study the pseudoline arrangement as a whole—its properties and results that surround it. Starting from Levi [6], pseudoline arrangements have been studied intensively with many interesting open questions [3]. As someone who likes combinatorial properties of curves, it is natural for me to think about

restriction of curves in a disk where all the bigons has been removed. In other words, this project is an excuse for me to finally read those papers that I always wanted to read.

Last, ideally I would like to make some actual progress on a problem, and based on previous experiences of working on final projects, I tend to choose a topic that is either too broad or too hard. So this time I would like to work on something more accessible and down-to-earth, and hopefully get to crack the problem somehow.

Plan of attack and half-baked ideas. I have not spend enough time think about it yet besides the obstruction I showed you last time when I tried to prove the statement with induction. The main issue is that I am not sure how to use the fact that all pairs of pseudolines cross *exactly* once.

One way to use the property is by creating a directed graph with all the bicolored intersections as nodes, and the edges points to a neighboring bicolored intersection guaranteed by the properties of minimal counter-example. However at this point I don't see a contradiction yet; maybe (just maybe) studying the graph would help.

It seems like if one makes assumptions on the colors of the pseudolines the problem becomes easier. Maybe identifying some subclass of pseudoline arrangements that has been studied before to work on would be more interesting.

I will probably spend some time reading surveys and papers on pseudoline arrangements first. It could be that during the process something more interesting would show up. As usual, aiming at a moving target :)

References

[1] Martin Aigner, Günter Ziegler, and Alfio Quarteroni. *Proofs from the Book*. vol. 274. Springer, 2010.

[2] Stefan Felsner. Bichromatic triangles in arrangements. Open Problems from the 33rd European Workshop on Computational Geometry, April 2017. ⟨http://csconferences.mah.se/eurocg2017/openproblems.pdf⟩.

[3] Stefan Felsner and Jacob E Goodman. Pseudoline arrangements. *Handbook of Discrete and Computational Geometry, ed., 3rd ed., CRC Press, Boca Raton, FL*, 125–157, 2017. Discrete Mathematics and Its Applications, CRC Press. ⟨https://www.csun.edu/~ctoth/Handbook/chap5.pdf⟩.

[4] Stefan Felsner and Alexander Pilz. Arrangements of approaching pseudo-lines. *Proc. 33rd European Workshop Comput. Geom.*, 229–232, 2017.

[5] Thomas A. Feo and J. Scott Provan. Delta-wye transformations and the efficient reduction of two-terminal planar graphs. *Oper. Res.* 41(3):572–582, 1993.

[6] Friedrich Levi. Die teilung der projektiven ebene durch gerade oder pseudogerade. *Ber. Math.-Phys. Kl. Sächs. Akad. Wiss. Leipzig* 78:256–267, 1926.

Computing shortest path trees in strongly connected planar digraphs with non-negative weights and prior knowledge

1 Problem

Given a strongly connected planar digraph $G = (V, D, rev, head)$, (i.e a planar digraph such that for any $u, v \in V$ there exists a path from u to v) with non-negative weight function $\omega : D \to \mathbb{R}^+$, and a set of shortest-path spanning trees $T_1, T_2, ..., T_k$, rooted at $v_1, v_2, ...v_k \in V$ respectively, how quickly can we compute the shortest path tree T rooted at a vertex in $v \in V$? There are several useful properties that can be used to improve upon running Dijkstra's algorithm.

1. We immediately know that the branching rooted at v in an arbitrary $T_i, 1 \leqslant i \leqslant k$ belongs to the optimal T.
2. Given a shortest path from v to u, we can upper bound the distance from v to w if w was a child of u in a T_i using
$$d(v, w) \leqslant d(v, u) + d(u, w)$$
3. If a subset of the shortest-path spanning tree (plus the outer face of the original graph) divides the graph into multiple faces then an interior vertex in one face cannot be an ancestor of an interior vertex in another face

I believe this problem could give insight into simple linear-space preprocessing for shortest path algorithms. Interesting related problems also exist. For instance, given a probability distribution $\pi : V \times V \to [0, 1]$, and $k \in \mathbb{N}$, find the optimal precomputed shortest path spanning trees to minimize expected computation time for computing the shortest path from v to u drawn from π.

2 Related Work

Frederickson found a planar single-source shortest path algorithm in $O(n\sqrt{\log n})$ time using r-divisions [1]. Henzinger et al. found a linear-time algorithm using recursive r-divisions, followed by running Dijkstra's algorithm in each division taking into account the size of each division [3]. In 2013, Klein et al. found

a recursive r-divison algorithm that uses linear space [4]. However, I have not seen this implemented on github or elsewhere.

Feueurstein and Marchetti-Spaccamela discussed an $O(n^2)$ shortest-path tree algorithm based on Frederickson's r-divisions [2]. In addition, they discuss an $O(n)$ space data structure based on all-pairs shortest distances between boundary vertices, which allows shortest paths to be computed in $O(n\sqrt{\log \log n})$ time.

References

[1] Greg N Federickson. "Fast algorithms for shortest paths in planar graphs, with applications". In: *SIAM Journal on Computing* 16.6 (1987), pp. 1004–1022.

[2] Esteban Feuerstein and Alberto Marchetti-Spaccamela. "Dynamic algorithms for shortest paths in planar graphs". In: *Theoretical Computer Science* 116.2 (1993), pp. 359–371.

[3] Monika R Henzinger et al. "Faster shortest-path algorithms for planar graphs". In: *journal of computer and system sciences* 55.1 (1997), pp. 3–23.

[4] Philip N Klein, Shay Mozes, and Christian Sommer. "Structured recursive separator decompositions for planar graphs in linear time". In: *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. ACM. 2013, pp. 505–514.

An Evaluation of Algorithms Exploiting Planar Structures In Image Segmentation

Jing Huang

Images can be naturally modeled as undirected planar graphs where each pixel represents a node. Since real-world images captured by cameras tend to be smooth, with neighboring pixels sharing similar intensity, connections of each node mainly lie between the direct neighboring pixels and the pixel itself. Edges representing such connections do not cross each other. The exact modeling of the image graph varies in different algorithms, yet the planarity provided by smoothness tends to be preserved and becomes useful in many image segmentation algorithms.

Image segmentation aims to partition pixels into groups to represent meaningful entities. The problem can be formulated as constructing a function $f : P \to L$ that assigns label l from a label set L to each pixel $p \in P$[1]. As the label set size the increases, the complexity increases. In the case where $|L| = 2$, the segmentation identifies foreground and background pixels. This particular problem can be reduced to a Min-Cut/Max-Flow problem by adding one source node connecting a pixel in the foreground and one sink node connecting a set of pixels in the background that forms a cycle winding around the source pixel exactly once. The weight on each edge is a function of the intensity gradient of two endpoints, penalizing low gradient. Schmidt et al[2]. proposed an $O(NlogN)$ segmentation algorithm built on the $O(NlogN)$ Max-Flow algorithm of Borradaile[3] where N is the number of pixels. The algorithm always augments the leftmost unsaturated path and tracks available paths effectively through spanning trees. As Schmidt's algorithm is one of the fastest segmentation algorithms for single source and sink, it will be the first candidate for the evaluation.

For $|L| > 2$, the vertex and weighted edge construction from the 2-label case can still apply. In practice, people solve multi-label segmentation as a Markov Random Field(MRF) using standard linear programming techniques, ignoring the planarity of the image graph. However, planarity still matters. Yarkony purposed a Planar Sub-Problem solver using the MRF framework but tighten the lower bound of linear programming relaxation for planar MRF[4]. As this is one of the few algorithms consider the planarity in the popular MRF approach, it will be the second candidate for evaluation.

A more topological approach for multi-label segmentation is given by Kropatsch et al.[5]. Kropatsch proposed building an irregular image pyramid based on the duality of planar graph. The bottom level C_1 is the original grid graph, while the dual image pyramid has the dual of original grid graph as bottom level C_1'. For each level above, the vertices of C_{k+1} is a subset of vertices from C_k. To obtain C_{k+1}, the algorithm grouping pixels in C_k by a primal edge contraction, which also corresponds to a dual edge removal. Following the primal edge contraction, there is a dual edge contraction to remove vertices with degree smaller than 2 in the dual graph. The contraction is applied to the edge with smallest weight at each vertices, where the weight measures similarity between vertices. Applying those two contractions yields a set of surviving vertices and non-surviving edges, whose connected components are spanning trees. When an edge is contracted, the connected component it belongs to is also contracted into a vertex. This construction is applying Boruvka's weighted minimum spanning tree algorithm on the dual image graph. As this algorithm uses a more complicated graph modeling than the common grid graph, it is interesting to inspect the geometrical properties of the segmentation and compare the accuracy with grid-graph based algorithms.

A major contribution of this project will be an experimental evaluation of the three planar graph algorithms mentioned above for image segmentation. Schmidt's Min-Cut/Max-Flow algorithm already has an available implementation, so an implementation is needed for Yarkony's Planar Sub-Problem approach and Kropatsch's image pyramid algorithm. Evaluation of each implementation consists of two parts, the speed and accuracy. The metrics for speed contain both theoretical complexity and the actual run time on

real images graphs. The algorithm will be evaluated on the Planar Graph Cut Dataset [2], which contains images up to a few megapixels. The accuracy of the algorithm mainly based on recall of the boundary pixel compared with human labeled ground truth. This will be evaluated on the Berkeley BSD Benchmark[6], which contains a few hundred images under 500*500 pixels and their ground truth segmentation. As none of the above algorithms has been thoroughly evaluated on both benchmarks, it will be interesting to see the potential trade-off between speed and accuracy in image segmentation.

A stretch goal is to further improve one of the algorithms. The improvement could come from exploiting some special properties of the planar graph extracted from the images. As the common modeling is a 4-regular grid graph, it might provide more efficient way to keep track of augmenting path and find separators. It is also interesting to investigate if the distribution of weights based on pixel intensity makes certain algorithm terminate earlier than running on a planar graph with random weight distribution. The other possibility is apply existing planar algorithms other than Min-Cut/Max-Flow to the segmentation problem. Especially for the multi-label image segmentation, there is no pure geometric based solution using Multiway Cut algorithms for planar graph. This is either due to the efficiency of solving MRF or the loss of planarity when connecting terminals to the pixels. Applying planar Multiway Cut algorithms may require to come up with a new modeling the graph as done in the image pyramid approach. With the recent advance in algorithms for planar graphs, there could be another potential speed up in the image segmentation.

References

[1] Jon Kleinberg and Eva Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *Journal of the ACM*, 49(5):616–639, 2002.

[2] Frank R. Schmidt, Eno Toppe, and Daniel Cremers. Efficient planar graph cuts with applications in computer vision. In *Computer Vision and Pattern Recognition*. IEEE, 2009.

[3] Glencora Borradaile. *Exploiting Planarity for Network Flow and Connectivity Problems*. PhD thesis, Brown University, 5 2008.

[4] Julian Yarkony. *Planarity Matters: MAP Inference in Planar Markov Random Fields with Applications to Computer Vision*. PhD thesis, Univeristy OF California, Irvine, 2012.

[5] Walter G. Kropatsch, Yll Haxhimusa, and Adrian Ion. In Abraham Kandel, Horst Bunke, and Mark Last, editors, *Applied Graph Theory in Computer Vision and Pattern Recognition*. Springer-Verlag Berlin Heidelberg, 2007.

[6] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, 7 2001.

Shailpik Roy
Netid: sroy15

Project Proposal

I propose a creative project. The project would involve building a web application that would help in the visualization of graphs and their corresponding algorithms. This should help in understanding the material taught in the course in a new light. If possible, I would prefer having Ziwei Ba as a teammate for the project; she contributed to the original project idea. The project would focus on integrating several graph/ plotting libraries and create a standard for visualization of graph algorithms.

Some of the algorithms described in class can be pretty complex and hard to visualize mentally so the web app should go a long way towards bridging that gap and making it easier to understand concepts taught in class

Some of the existing graph visualization applications that I have encountered are https://visualgo.net/en and https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

Visualgo helps you visualize tree and graphs and algorithms like shortest path travelled, cycles etc.

The visualization algorithms for usfca are mostly oriented towards data structures like arrays and linked lists. It also has visualizations of Prims and Dijkstra's algorithms for graphs.

In order to implement our project, I have looked into the following libraries:

Graphviz: It's a python library that facilitates the creation and rendering of graph descriptions. Using this would make it possible to create a graph object, assemble the graph by adding nodes and edges. Thus, it would be very helpful in creating the graphs and dual graphs and flow networks and involved algorithms.

NetworkX: It's a python library for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. It has a lot of graph algorithms already integrated so it would be helpful for our project.

CS 598 Project Proposal

Qixin (Stark) Zhu

"Divide and conquer" is one of the oldest and most widely used techniques for designing efficient algorithms [1]. A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem [2]. This strategy can be successfully and efficiently applied to graph problems, provided we can quickly separate the graph into roughly equal subgraphs [1].

The separator theorem states that for any n-vertex planar graph G=(V, E) (n = |V|) and for any weight function w: V → $\mathbb{R}^+$, V can be partitioned into 3 sets A, B, S ⊆ V such that (1) A and B are α-balanced: w(A), w(B) ≤ $\alpha \cdot w(V)$ for some $\alpha \in (0,1)$; (2) A and B are separated: no edge joins a vertex in A with a vertex in B; (3) separator S is small: $|S| \leq f(n)$; (4) the partition can be found efficiently in linear time.

There are 2 naïve separators for planar graphs. The first one is the level separator, which constructs a BFS spanning tree and define levels of all vertices to be the distance from a fixed root vertex. It selects the set of vertices of the median level as a separator, which is balanced but may be big. The second one is the fundamental cycle separator, which construct a BFS spanning tree of the dual vertices. It selects the edge, which cut the dual tree into 2 balanced subtrees, as the edge to be added into the primal spanning tree, which then forms a cycle as the separator. But the cycle founded may also be big, since the depth of the spanning tree is not guaranteed to be small.

Lipton and Tarjan (1979) [3] combined the ideas from the naïve separators and constructed a balanced separator whose sized can be limited to $O(\sqrt{n})$. The high-level idea is to construct the BFS spanning tree and find the median level, same as the level separator method. Then find a level above and below the median level that contains $\sqrt{n}$ vertices. Uses these 2 levels together with 2 branches from a fundamental cycle separator to construct another separator of size at most $\sqrt{8n}$, aka $2.83\sqrt{n}$. The constant factor was reduced to $\sqrt{6}$, aka 2.45, by Djidjev (1982) using more careful analysis [4].

In this project, the 2 naïve separator will be implemented as well as the Lipton-Tarjan separator. They will be tested and compared on various kinds of planar graph, including both grids induced by graphics and planar graphs from actual world such as maps. The first goal in this part is to understand the Lipton-Tarjan algorithm through implementation. The second goal is to evaluate the performance of the theoretical guaranteed $O(\sqrt{n})$ bound in real practice. The third goal is to observe the size of the naïve separators by choosing the root of spanning tree at random and explore some heuristics in root selection.

One shortage of the Lipton-Tarjan separator is that the 2 levels of $\sqrt{n}$ vertices may not be connected by an edge, so the separator S is not necessarily connected. Though connectivity is not required, it may bring some convenience in usage, provided finding such connected separator takes

no extra effort and can be completed in the same time bound O(n). Miller (1986) [5] proved that a small simple cycle separator of size $\sqrt{8n}$ can be found in O(n) time. The constant factor was then improved by Djidjev & Venkatesan (1997) [6] to $1.97\sqrt{n}$.

In this project, the algorithm for finding small simple cycle separator will be implemented. The Miller separator will be compared with Lipton-Tarjan separator and the fundamental cycle separator based on various types of planar graphs as described above. The first goal is to understand Miller's separator and the proof of size bound. The second goal is to evaluate the performance and size difference between the theoretical guaranteed cycle separator and the naïve one.

To take one step further, the separators can be used recursively to form a separator hierarchy, namely r-division. An r-division of G is a decomposition into O(n/r) edge-disjoint pieces (called region), each of which has vertices less than or equal to r and has $O(\sqrt{r})$ boundary vertices. The root of the separator hierarchy tree is the entire graph itself, and the two children are the roots of separator trees constructed recursively for the subgraphs A and B induced by the root-level separator S. A naïve method of constructing this r-division, by Frederickson (1986) [7], is to apply the linear-time separator when traversing the separator tree, which will take a total of O(n logn) time. A second approach is to preprocess the graph with a ρ-clustering of size $\sqrt{r}$, and contract each piece into a single node, thus shrinking the graph into n'=O($n/\sqrt{r}$) vertices. Doing naïve recursion on this shrunk graph takes O($n/\sqrt{r}$ logn) time, which is governed by the time of expanding pieces back and doing another O(logr) levels of recursion. So the total time cost of this approach is O(n logr). The time bound of separator hierarchy construction can be further improved to linear by using appropriate data structures to perform the partitions, given by Goodrich (1995) [7].

In this project, the 3 different approaches of r-division will be implemented and compared. The constant factor associated with the linear time r-division will be studied for practical purpose. All separator algorithms mentioned above may be plugged into the r-division framework and compared for practice purpose. The application of r-division may be implemented and studied if time allowed, which may include the algorithms for planar graphs to solve the multiple source shortest path problem in O(n logn) time [9], the single source shortest path problem in O(n loglogn) time and O(n) time [10] and the maximum flow problem in O(n logn) time [11].

[1] Jeff's notes
[2] https://en.wikipedia.org/wiki/Divide_and_conquer_algorithm
[3] Lipton, Tarjan, A separator theorem for planar graphs, SIAM Journal on Applied Mathematics, 36 (2): 177–189, 1997
[4] Djidjev, On the problem of partitioning planar graphs, SIAM Journal on Algebraic and Discrete Methods, 3 (2): 229–240, 1982.
[5] Miller, Finding small simple cycle separators for 2-connected planar graphs, Journal of Computer and System Sciences, 32 (3): 265–279, 1986
[6] Djidjev, Venkatesan, Reduced constants for simple cycle graph separation, Acta Informatica, 34 (3): 231–243, 1997.

[7] Frederickson, Fast algorithms for shortest paths in planar graphs, with applications, SIAM J. Computing, 1004-1022, 1987.
[8] Goodrich, Planar separators and parallel polygon triangulation, J. Comput. System Sci. 51, 374–389, 1995.
[9] Klein, Multiple-source shortest paths in planar graphs, Proceedings of 16th ACM-SIAM Symposium on Discrete Algorithms, 146-155, 2005
[10] Klein, Rao, Rauch, Subramanian, Faster shortest-path algorithms for planar graphs, Journal of Computer and System Sciences, 55 (1): 3–23, 1997
[11] Erickson. Maximum Flows and Parametric Shortest Paths in Planar Graphs. Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, 794-804, 2010.

Ziwei Ba

Shailpik Roy (Teammate)

CS598JGE- Computational Topology

November 2, 2017

Final Project Proposal

General Idea

We are aiming to make a web app using some existing libraries to visualize algorithms for graphs and curves that we have learned in class. Features of our app will include live animation of algorithms, custom data input to generate the curve, and an explanation of each algorithm used. We're not looking to make a graph theory library from scratch- rather, utilize existing tools to create our app to visualize some definitions and algorithms we learned in class.

Motivation

A lot of things related to graphs are difficult to conceptualize until you work out an example by hand. When I watch the lecture videos, I have to pause every time a new topic is discussed to work out a concrete example, and apply whatever theorem or algorithm we just discussed. As a result, I usually take a day or two to catch up on the past week's lectures. A lot of this time could be avoided with a program or an app that could do the visualization for me, so I decided to create one. This would help students learn about these algorithms, and also help non-graphing people become more interested in graphs with a fun visualization app.

Existing Technologies

Inspiration for this project came from tree visualization apps that I discovered when I was looking for tools to help teach my students in CS225 AVL trees.

Tree Visualization

https://www.cs.usfca.edu/~galles/visualization/

Currently only btrees and avl trees work here, but Professor Galles from University of San Francisco made some neat interactive tree visualization apps with the Java Swing library. Users can add nodes, delete nodes, and most importantly increase or decrease animation speed. This really helped my 225 students grasp trees a lot better. Our app will be similar to this in that we focus on a few data structures, but we will accompany each visualization with a simple explanation of the algorithm or definition.

VisuAlgo

https://visualgo.net/en

VisuAlgo is a database of algorithms visuals- trees, graphs, MSTs, etc. The interface is more code-like than the tree visualization app, where user can input arguments to functions. As the

animation plays, there is a little popup window on the side displaying the current line of code that is executing. The distinguishing feature of VisuAlgo is the large number of algorithms its database contains. We will be focusing on a small number of graph definitions and structures that are helpful to visualize, such as curve and path definitions, and homotopy definitions for different structures.

Implementation Ideas

Java Swing Library

Galles (from the USFCA page) has a helpful tutorial to add more algorithms in Java Swing. I haven't looked into this option much, but his library seems pretty comprehensive. Java Swing is a pain to deal with though, so I would rather hold off on this option.

Cytoscape

Cytoscape is a JavaScript graph visualization library and our most likely option at this point. It has an intuitive API and many options to customize colors and graph layout. I haven't looked into animation with Cytoscape yet, but so far it looks like a good implementation option.

Dracula

Dracula looks a lot more lightweight and easy to use than Cytoscape. Currently there seems not to be an option to animate a graph, so we may have to modify the source code for that ourselves.

A Survey on Topological Music Analysis (Proposal)

Haizi Yu

Music plays a ubiquitous and inseparable role in our lives. However, as an art form, music is also a most mysterious and fascinating object that not only allows numerous ways in its appreciation, but also offers endless possibilities in its creation. For centuries, music theory and its development has brought more and more tools and methodologies for people to describe, to understand, and further to make music. In the recent decades, the line between music and mathematics has been increasingly blurred, creating a somewhat surprising fusion between art and science: modern music theorists have begun to adopt more and more mathematical approaches to reinterpret classic theories/pieces as well as to suggest new compositional possibilities. The discovered math behind music not only makes many musical intuitions precise, but more importantly positions what we know about music in a bigger and self-contained picture, generalizing the same set of principles to many unexplored music territories and exposing new dimensions for music research.

Among many of the mathematical perspectives, this survey paper focuses on the topological/geometrical methods on music analysis. A topological/geometrical view of music allows us to better visualize and identify musical activities that are otherwise hard to perceive either by listening to soundtracks or by sight-reading from sheet music. The benefits obtained from this new dimension of music perception are not purely for the purpose of better understanding or happiness gained from satisfying one's curiosity, but more importantly give rise to new techniques that can both revitalize (or complete) classical works in the human history and inspire modern music composition. The goal of this paper is to make a comprehensive survey on a large corpus of existing and ongoing work regarding a topological/geometrical view of music, as well as to suggest some general approaches in topological data analysis and computational topology that are potentially interesting and applicable in music.

The main body of this survey paper divides into three parts. The first part is intended to give a topological/geometrical overview of contemporary/modern music theory. This will begin with an introduction on various music data representations, a handful of important music terms and concepts (such as musical pitch, scale, chord, voice leading), as well as several basic mathematical tools in modeling musical activities. We'll then formalize these music notions by providing their mathematical definitions in which a major part will include basic topics like pitch class, set class, melodic graphs and paths, random walks, as well as more advanced topics built on these aforementioned basics, e.g. equivalence relations and their induced quotient/covering spaces: multi-dimensional manifolds and orbifolds. After that, we will introduce metrics (the very important notion of measuring similarity or distance) on the proposed musical space, and represent musical activities as curves/paths/graphs on various musical surfaces. We will conclude this part by providing some examples to illustrate why these topological/geometrical views of musical notations and activities are illuminating in both revisiting the classical tonal theory and inventing new possibilities in contemporary/modern music composition (for instance, we may include some related work on computer music in this university that were benefitted from the topological/geometrical interpretations).

The second part is intended to jump out of the domain of music, and present an overview of topological data analysis in general. This part will begin with an introduction on positioning topological data analysis in a bigger picture, pointing out its relation to many other areas in the field of data science, e.g. machine learning, statistical inference, etc. Comparisons will be made to these interconnected areas, in which we aim for a summary that elicits pros and cons of each when applied to different domains and used for different purposes. In particular, music will be the case study here to illustrate the point. Instead of a comprehensive overview that touches every corner of topological data analysis, a larger emphasis will be placed on techniques that demonstrate potential connections to music. So content in this part will be mostly presented as a two-part entity, in which an introduction of some topic will be presented first, and its connection to music will be discussed in the second. Some potential topics will include but not limited to persistence theory, topological inference, homology, clustering, as well as some new trends in topological data analysis. We will also cover some state-of-the-art works from some of the leading research groups in this area, including but not limited to (and subject to change) Tamal Dey's, John Harer's, Herbert Edelsbrunner's, Gunnar Carlson's groups.

The third part is intended to talk about both existing and potential applications that could be spawned from the development of topological data analysis. This part will be further organized into two sub-parts. The first sub-part will introduce known applications in music, including but not limited to modern music analysis, automatic music composition and pedagogy. Again, we will cover some state-of-the-art works from some of the leading music research groups, including but not limited to the music theory and composition division in this university, CCRMA at Stanford, and some of composer Dmitry Tymoczko's latest work. The second sub-part will introduce known applications in other domains but thought be potentially similar or closely related to music. Two popular applications are possibly researches in the visual domain such as shape study, graph reconstruction, image analysis, as well as researches in network analysis like metric, clustering, and persistence homology of a network. We may mention some cool relevant applications in biological data analysis (bioinformatics) as well. But again, this is not a mere showcase of various applications: every mentioned application will be discussed with its (potential) connection to music. We will conclude this part by suggesting general takeaways that might be useful to researchers who are interested in the interdisciplinary area of music and computational topology, as well as challenges that one might need be aware of.

Topological Data Analysis: A Survey

Kevin Hong
khong18

1 Background

When approaching the problem of extracting information and making predictions on data, the common approach is one of any number of machine learning algorithms, and this field has only exploded in recent years. However, machine learning faces challenges in dimensionality, noise, and the need to use structured metrics in order to create a tractable problem. Recently, a new method for analysis has appeared, in the form of Topological Data Analysis. It serves as a robust complement to machine learning, utilizing techniques from topology and beyond to draw conclusions about the qualities of data presented.

2 Motivation

Our first instinct when seeking to understand data presented to us is to create a model that best fits the data, leveraging the quantitative components of input data to computationally derive this model. While a model may be present, the shape of the data is also incredibly important to understand the underlying structuring of the data. An example given by Epstein et al.[Epstein et al.(2011)Epstein, Carlsson and Edelsbrunner] is that of the Lotka Volterra Equation. If we plotted data surrounding an ecological system of predators and prey, namely populations at different times, we would observe a simple closed curve in three dimensions, giving the seemingly obvious but important conclusion that these populations are periodic in nature.

3 Proposal

In my survey of Topological Data Analysis (TDA), I hope to explore the following areas:

3.1 Underlying Mathematical Foundations

Input to TDA is in the form of point clouds, a finite set of points with a well defined measure of distance between any pair of members in the set. Once obtained, shapes and structures are built upon the data traditionally taking the form of simplices, abstractions of the notion of triangles to arbitrary dimensions. A set of simplices is known as a simplicial complex, and those can be nested to represent the final structure of the data. Further explorations can be done in covers of points, abstract simplicial complexes and their special cases, such as the Vietoris-Rips complex, and the ech complex.

3.2 Applications of the Structure

Various applications of the now structured data include visualization as characterized by Singh et al [2] and feature extraction through persistent homologies, and its representations as persistence barcodes or diagrams. [Edelsbrunner et al.(2002)Edelsbrunner, Letscher and Zomorodian] It has also been shown that persistent homologies are resistant to noise [Cohen-Steiner et al.(2007)Cohen-Steiner, Edelsbrunner and Harer].

3.3 Applications of TDA

Topological Data analysis has been applied successfully to fields such as materials science where structure has been extracted from amorphous materials [Nakamura et al.(2015)Nakamura, Hiraoka, Hirata, Escolar and Nishiura], patterns of viral evolution [Chan et al.(2013)Chan, Carlsson and Rabadan], disease progression analysis [Nicolau et al.(2011)Nicolau, Levine and Carlsson], and more.

3.4 Relationship with Machine Learning

Features extracted using persistent homologies have been used as inputs to novel machine learning algorithms such as Sparse-TDA [Guo et al.(2017)Guo, Manohar, Brunton and Banerjee], and problems such as object recognition [Li et al.(2014)Li, Ovsjanikov, Chazal].

References

[Chan et al.(2013)Chan, Carlsson and Rabadan] CHAN, J. M., CARLSSON, G. and RABADAN, R. (2013). Topology of viral evolution. *Proceedings of the National Academy of Sciences* **110** 18566–18571.
URL http://www.pnas.org/content/110/46/18566.abstract

[Chazal and Michel(2017)] CHAZAL, F. and MICHEL, B. (2017). An introduction to topological data analysis: fundamental and practical aspects for data scientists.

[Cohen-Steiner et al.(2007)Cohen-Steiner, Edelsbrunner and Harer] COHEN-STEINER, D., EDELSBRUNNER, H. and HARER, J. (2007). Stability of persistence diagrams. *Discrete & Computational Geometry* **37** 103–120.
URL https://doi.org/10.1007/s00454-006-1276-5

[Edelsbrunner et al.(2002)Edelsbrunner, Letscher and Zomorodian] EDELSBRUNNER, LETSCHER and ZOMORODIAN (2002). Topological persistence and simplification. *Discrete & Computational Geometry* **28** 511–533.
URL https://doi.org/10.1007/s00454-002-2885-2

[Epstein et al.(2011)Epstein, Carlsson and Edelsbrunner] EPSTEIN, C., CARLSSON, G. and EDELSBRUNNER, H. (2011). Topological data analysis. *Inverse Problems* **27** 120201.
URL http://stacks.iop.org/0266-5611/27/i=12/a=120201

[Guo et al.(2017)Guo, Manohar, Brunton and Banerjee] GUO, W., MANOHAR, K., BRUNTON, S. L. and BANERJEE, A. G. (2017). Sparse-tda: Sparse realization of topological data analysis for multi-way classification.

[Nakamura et al.(2015)Nakamura, Hiraoka, Hirata, Escolar and Nishiura] NAKAMURA, T., HIRAOKA, Y., HIRATA, A., ESCOLAR, E. G. and NISHIURA, Y. (2015). Persistent homology and many-body atomic structure for medium-range order in the glass.

[Nicolau et al.(2011)Nicolau, Levine and Carlsson] NICOLAU, M., LEVINE, A. J. and CARLSSON, G. (2011). Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *Proceedings of the National Academy of Sciences* **108** 7265–7270.
URL http://www.pnas.org/content/108/17/7265.abstract

[Singh et al.(2007)Singh, Mmoli, Carlsson] Singh, G., Mmoli, F., and Carlsson, G. E. (2007). Topological methods for the analysis of high dimensional data sets and 3d object recognition. In SPBG, pages 91100. Citeseer.

[Li et al.(2014)Li, Ovsjanikov, Chazal] Li, C., Ovsjanikov, M., and Chazal, F. (2014). Persistence-based structural recognition. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pages 2003 2010.

1 Project Summary

The goal of this project is to do a comprehensive survey on the different methods for deciding the planarity of an arbitrary graph in order to help others gain intuition into how such algorithms work. This would involve writing a set of lecture notes covering topics including a history of the development of these algorithms, some practical applications of finding planar graphs, and detailed examples.

2 Current Status

The first linear time algorithm for testing the planarity of a graph was introduced in 1974 by Hopcroft and Tarjan. However, the original algorithm is complex and difficult to implement, so there have been numerous attempts over the years to simplify their approach. This algorithm is referred to as the "path-addition algorithm".

Another algorithm called the "vertex addition algorithm" was developed by Lempel, Even, and Cederbaum in 1966, which is based on considering the vertices of the graph one at a time.

Finally, an algorithm based on the same principles as the left-right planarity test for unsigned Gauss codes was developed by Fraysseix and Rosenstiehl. According to a 2004 paper that compared a number of different algorithms for this problem, this algorithm has been implemented very efficiently, and a later 2009 paper gave a full description of the algorithm.

3 Relevant Resources to Investigate Further

1. Wikipedia page on planarity testing: Link
2. "Efficient Planarity Testing"
   This is the 1974 paper by Hopcroft and Tarjan detailing the original linear time planarity testing algorithm. Link
3. "Stop Minding Your Ps and Qs: Implementing a Fast and Simple DFS-Based Planarity Testing and Embedding Algorithm"

The 2004 study that compares 6 different algorithms for solving this problem. Link

4. "A new planarity test"
   An attempt at simplifying the approaches of Hopcroft/Tarjan and Booth/Lueker. Link

5. "The Left-Right Planarity Test"
   A full description of Rosenstiehl and de Fraysseix's left-right characterization of planar graphs. Link

Clustering and Planar Graphs

Ross Vasko

November 13, 2017

1 Introduction

I will be attempting to complete a survey over clustering in the case of planar graphs. It does not appear that a significant amount of work has been done on clustering for planar graphs, from an initial review of materials. Only a few types of clustering have been examined for the planar case. Primarily, it seems that problems related to finding a "dense" subgraph, correlation clustering, and (k, r)-centers have been the most investigated.

The remainder of this report will be a brief overview of approaches and papers that I will investigate further. I will look to further understand and summarize the following results, as well as attempt to look for additional promising approaches to improve graph clustering in the planar case.

2 "Dense" Subgraphs

It appears that one of the first works done in clustering planar graphs is a result that shows finding a connected subgraph that is "dense" enough is NP-complete [4]. Keil and Brecht consider *h-cluster* problem. In this problem, the input is a graph G, and two integers h and j and it is asked "are there h vertices of G that induce a subgraph with at least j edges?". The connected variant of this problem requires that this induced subgraph be connected as well. The authors show that the connected variant problem is NP-complete in the planar case by a reduction from the *connected vertex cover* problem on graphs which are planar and have a maximum degree of four.

3 Correlation Clustering

A survey [3] on correlation graph clustering points to two results ([7] and [5]) specific to the planar case for the *weighted min graph correlation clustering (WMCC)* problem. WMCC takes a graph with edges labeled with a + or a -, which indicate the similarity of the vertices incident on the edges. WMCC asks us to find a partition of the vertices that minimizes the dissimilarity in the partitions [5].

Voice et al. [7] give a result that shows that WMCC is NP-hard for planar graphs.

Klein et al. [5] show that the WMCC problem can be reduced to the *two-edge-connected augmentation* problem. This allows the authors to obtain a PTAS for WMCC. The claimed motivation for considering the planar case in this problem is for image segmentation.

Another work [8] actually performs image segmentation considering the WMCC problem. Their approach includes using weighted perfect matchings.

In another paper on general correlation clustering by Demaine et al. [1] it is noted several times that the max-flow min-multicut ratio result by Tardos and Vazirani [6], allows for constant factor approximations in the work of [1] for the planar graph case. This observation encourages me to investigate other common graph clustering algorithms and to see if any of the approaches and algorithms used in these might be easier in the planar case. As in the above case, this could improve the algorithm as a whole.

4 (k, r)-centers

Graph clustering based on (k, r)-centers are also classic problems in computer science. The *(k, r)-centers* problem asks if there exist k vertices in a graph G such that every vertex of G is a distance at most r from the selected k vertices. This problem is shown to be fixed-parameter tractable by Demaine et al. [2] and runs in time $f(k, r)n^{O(1)}$.

References

[1] Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2-3):172–187, sep 2006.

[2] Erik D Demaine, Fedor V Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Transactions on Algorithms (TALG)*, 1(1):33–47, 2005.

[3] Victor Il'ev, Svetlana Il'eva, and Alexander Kononov. Short survey on graph correlation clustering with minimization criteria. In *Discrete Optimization and Operations Research*, pages 25–36. Springer International Publishing, 2016.

[4] J Mark Keil and Timothy B Brecht. The complexity of clustering in planar graphs. *J. Combinatorial Mathematics and Combinatorial Computing*, 9:155–159, 1991.

[5] Philip N. Klein, Claire Mathieu, and Hang Zhou. Correlation Clustering and Two-edge-connected Augmentation for Planar Graphs. In Ernst W.

Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 554–567, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[6] Éva Tardos and Vijay V. Vazirani. Improved bounds for the max-flow min-multicut ratio for planar and kr, r-free graphs. *Information Processing Letters*, 47(2):77–80, aug 1993.

[7] Thomas Voice, Maria Polukarov, and Nicholas R Jennings. Coalition structure generation over graphs. *Journal of Artificial Intelligence Research*, 45:165–196, 2012.

[8] Julian Yarkony, Alexander Ihler, and Charless C. Fowlkes. Fast planar correlation clustering for image segmentation. In *Computer Vision – ECCV 2012*, pages 568–581. Springer Berlin Heidelberg, 2012.

IMAGE PROCESSING WITH MAXIMUM FLOW

YASHA MOSTOFI

1. Background

The maximum flow problem is commonly applied to problems involving image processing and computer vision. Most of the problems that are easily solvable are "low-level" problems (called "early vision"). Such problems include image smoothing, the stereo correspondence problem, and image segmentation. However, there are limits and drawbacks to using max-flow/min-cut for these problems. Some criticism found in literature include artifacts from representing the image with a 4-connected lattice, shrinking bias (not working well with smaller objects in the image), only supporting binary labeling, and large memory usage as the problem size increases. Newer literature is exploring different approaches to using max flow (with both major and minor modifications) to solve problems related to image processing, with varied results.

2. Proposal

I propose to survey and aggregate three different areas related to maximum flow and minimum cut.

2.1. **Successes.** I want to explore current problems that are solvable with standard maximum flow algorithms and produce well-defined results. That is, the solution obtained using maximum flow is well-defined and clear of most criticisms present in literature. From initial (and minimum) research, it appears that this is almost entirely image segmentation on rectangular images. As part of this, I want to also present the canonical way of converting a given image to a graph for maximum flow, and what sort of underlying data structures perform better than others for this specific problem.

2.2. **Boundaries.** In addition to surveying the successes, I wish to explore the boundaries of image processing with maximum flow (with minimum deviation from the standard algorithms). More specifically, where does maximum flow fail, and how are researchers attempting to remedy the issues? What modifications to the maximum flow algorithm are necessary to be able to solve such problems? Is there a certain class of image that cannot be processed with maximum flow?

Date: November 10th, 2017.

Are there any pre-processing steps needed to perform on the input image(s) before applying maximum flow?

2.3. **New areas.** Finally, I'm curious to see what new types of problems related to image processing and computer vision are being tackled using maximum flow-based approaches. More specifically, I'm interested to see if image depth information can be determined using one or more image from varying angles and performing some sort of maximum flow processing on the images. I'm not sure if such research is being done, but I'm fairly confident that obtaining depth information given a single image is a problem currently being worked on given industry trends. In addition, I'm eager to see if image processing with maximum flow has opened up research in other applications of maximum flow due to potential similarities.

References

[1] Jing Yuan, Egil Bae, Xue-Cheng Tai, Yuri Boykov, *A Study on Continuous Max-Flow and Min-Cut Approaches*
[2] Wikipedia, *Graph cuts in computer vision*
[3] Jianbo Shi, David Martin, Charless Fowlkes, Eitan Sharon, *Tutorial Graph Based Image Segmentation*
[4] Ross Whitaker, *Graph Cuts Approach to the Problems of Image Segmentation*