Context-Free Path Queries, Planar Graphs and Friends

Alexandra Istomina
Saint-Petersburg State University
Saint-Petersburg, Russia

Alexandra Olemskaya
Inria Paris-Rocquencourt
Rocquencourt, France

Ekaterina Shemetova
Inria Paris-Rocquencourt
Rocquencourt, France

Semyon Grigorev
Rajiv Gandhi University
Doimukh, Arunachal Pradesh, India

ABSTRACT

Abstract is very abstract.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ACM Reference Format:
Alexandra Istomina, Alexandra Olemskaya, Ekaterina Shemetova, and Semyon Grigorev. 2018. Context-Free Path Queries, Planar Graphs and Friends. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Context-Free Path Querying (CFPQ) is a sublcass of Language-constrained path problem, where language is set to be Context-Free.

Importance of CFPQ. Application areas. RDF, Graph database querying, Graph Segmentation in Data provenance, Biological data analysis, static code analysis.

1.1 An Example

Example of graph and query. Should be used in explataion below.

1.2 Existing CFPQ Algorithms

Number of problem-specific solutions in static code analysis.

Hellings, Ciro et al, Kujpers, Sevon, Verbitskaya, Azimov, Ragozina

1.3 Existing Theoretical Results

Existing theoretical results

Linear input. Valiant [12], Lee [10].

Yannacacis [13]? Reps?

Bradford [1]

RSM [4].

C alias analysis [14]

Chatterjee [3]

For trees

Truly-subcubic for Language Editing Distance [2].

Truly-subcubic algorithm is stil an open problem.

1.4 Our Contribution

This paper is organized as follows. !!!!!

2 PRELIMINARIES

We introduce !!!!

2.1 Context-Free Path Querying

Graph, grammar, etc.

Let $i\pi j$ denote a unique path between nodes i and j of the graph and $l(\pi)$ denotes a unique string which is obtained from the concatenation of edge labels along the path π. For a context-free grammar $G = (\Sigma, N, P, S)$ and directed labelled graph $D = (Q, \Sigma, \delta)$, a triple (A, i, j) is *realizable* iff there is a path $i\pi j$ such that nonterminal $A \in N$ derives $l(\pi)$.

2.2 Tensor-Based algorithm for CFPQ

Algorithm 1 Kronecker product context-free recognizer for graphs

```
1: function CONTEXTFREEPATHQUERYING(D, G)
```

2.3 Planar Graphs

A planar graph $G = (V, E)$ is a graph that can be embedded in the plane.

Outer face - unbounded face in specific embedding.

Directed graph (*digraph*)

...

2.4 Dynamic reachability algorithms

We consider algorithms that solve the problem of reachability in planar directed graphs. In the *dynamic reachability problem* we are given a graph G subject to edge updates (insertions or deletions) and the goal is to design a data structure that would allow answering queries about the existence of a path.

We need to answer the queries of type: "Is there a directed path from u to v in G?". If vertex u in all the queries is fixed we say that algorithm is *single-source*. It is said to be *all-pairs* if vertices u, v can be any vertices of planar digraph G, in this case it can be also called *dynamic transitive closure.*

We say that the algorithm is *fully dynamic* if it supports both additions and deletions of edges. It is said to be *semi dynamic* if it supports only one of these updates. If semi dynamic algorithm supports additions only it is called *incremental*, if deletions only - *decremental.*

3 CFPQ ON PLANAR DIGRAPHS

3.1 Dynamic transitive closure

We want to consider algorithms for semi-dynamic transitive closure for the case of a planar graph. Our algorithm needs to support only edge insertions that do not violate planarity.

There are several algorithms that look into the problem of dynamic reachability in planar digraphs and have sublinear both update and query time [11], [9], [8].

Some algorithms [5], [9] restrict insertions to only that edges that respect the specific embedding of the planar graph.

maybe we can choose one embedding at random and hope that nothing will violate it. So it will be monte-carlo (why it will have good probability? where from can we get all the possible non-isomorphic embeddings?)

The algorithms that allow edge insertions not with respect to the embedding of the graph, but to the planarity, are described in [11], [7]. The main idea of both articles is the same: a planar graph is separated into *clusters* (edge induced subgraphs) for which reachability is shown by their sparse substitute S. *Sparse substitute* is a graph that contains the reachability information between some set of vertices that lie in the same face of an embedded graph.

We show the results of [11] more closely as they differ from the results of [7] only by a logarithmic factor, have the same idea and are easier for understanding.

Updates and queries are the following: adding or deleting the edge between vertices u, v, checking reachability, checking if new edge (u, v) violates planarity.

Every update procedure modifies a constant number of clusters, for which we reconstruct their sparse substitutes. If the procedure added the edge, new edge forms its own cluster. Moreover after every $O(n^{1/3})$ insertions we rebuild cluster partition and sparse substitutes from the scratch so that clusters do not lose their properties, this rebuilding time is distributed between these $O(n^{1/3})$ insertions, so insertion time is amortized.

After splitting graph into clusters and looking into its sparse substitute update or query about vertices u, v can be done by placing clusters of the vertices u, v in the original graph G into the graph of sparse substitutes S.

For maintaining dynamic transitive closure in this way we only need planarity to divide graph G into clusters and for each cluster to build its sparse substitute. As we rebuild cluster partition and substitutes at the beginning and after every $O(n^{\frac{1}{3}})$ edge-insertions, graph needs to remain planar during the updates.

The results are the following.

(1) The amount of time for preprocessing is $O(n \log n)$ — in this time we find the separation of the given graph G into $O(n^{\frac{1}{3}})$ clusters (each consists of no more than $n^{\frac{2}{3}}$ edges of G) and build sparse substitution graph S of total size $O(n^{\frac{2}{3}} \log n)$ for them.
(2) After that we can perform add operation in $O(n^{\frac{2}{3}} \log n)$ amortized time, delete operation in $O(n^{\frac{2}{3}} \log n)$ worst-case time.
(3) Reachability query takes $O(n^{\frac{2}{3}} \log n)$ worst-case time.
(4) Checking if the graph is planar can be done in $O(n^{\frac{1}{2}})$ amortized time.

Let us look closely on how can we use the power of sparse substitution on our incremental transitive closure problem.

Proposition 3.1. *We can maintain the structure described in [11] not only for planar graphs, but for planar graphs with $O(n^{\frac{1}{3}})$ edges that violate planarity.*

□ In [11] the number of clusters is $O(n^{\frac{1}{3}})$. In the beginning and after every $O(n^{\frac{1}{3}})$ edge insertions we rebuild sparse substitute graph S. and in these moments we need planarity. We can additionally maintain the set of non-planar edges, each edge of this set will present its own cluster and will not take part in the rebuilding of sparse substitution. Edge is *non-planar* if its insertion to current planar graph G makes it non-planar. ■

Proposition 3.2. *We can add edges in the graph and print out new pairs of reachable vertices for every insertion in $O(n^{\frac{5}{3}})$ total time for every sequence of insertions if our model allows parallel computations.*

□ From [11] we can add an edge (u, v) in $O(n^{\frac{2}{3}} \log n)$ amortized time. We want to get all pairs of vertices that are connected through the new edge.

We can run DFS from v and DFS on reversed edges from u in graph S of sparse substitutes. DFS runs in time proportional to the size of the graph $S - O(n^{\frac{2}{3}} \log n)$. After that for every cluster in the original graph G we create dummy vertex s and edges from s to all boundary vertices in the cluster that were reached from u and v respectively. Then run DFS from this dummy vertex s. We print out every vertex that was reached by our DFSs, so we get two lists U and V for beginnings and the endings of the paths, going through (u, v).

If any vertex w is reachable from v (without loss of generation), then it is a boundary vertex or lie in some cluster and is reachable from some boundary vertex of this cluster. In both cases, one of DFS's will print it out and the algorithm is correct.

If we can run these DFS's in parallel (there are $O(n^{\frac{1}{3}})$ clusters and the same number of DFS's), then each of them will take $O(n^{\frac{2}{3}})$ amount of time (all cluster have $O(n^{\frac{2}{3}})$ edges by definition in [11]). As maximum number of edges in the planar graph is linear, the total amount of time will be $O(n^{\frac{5}{3}})$ for every sequence of edge insertions. ■

Conjecture 3.3. *If our model does not allow parallel computations total amount of time for every sequence of insertions is at most $O(n^2)$ and planarity does not get us any advantage in solving the problem of dynamic reachability (in amortized time per edge and query).*

[Thoughts] If our model can not run algorithms in parallel, then the amount of time taken by iterating DFS's described above for every cluster is linear — $O(n)$. This means that usual DFS on the original graph G has the same complexity and we will spend in total $O(n^2)$ time. ■

maybe we can spare some space ($O(n^2)$?) so we can store list of reachable vertices from any boundary vertex and somehow update them during the edge additions?

3.2 Planarity of Kronecker product of digraphs with labels

Planarity of Kronecker product. From results of [6] we know criteria for planarity of Kronecker product of two graphs:

THEOREM 3.4 ([6]). *Let G_1 and G_2 be connected graphs with more than four vertices. Then $G_1 \wedge G_2$ is planar if and only if either:*

(i) *one of the graphs is a path and the other one is 1-contractible to a path or a circuit, or*
(ii) *one of them is a circuit and the other is 1-contractible to a path.*

A 1-contraction of G is the removal from G of each vertex of degree 1 (and its incident edge).

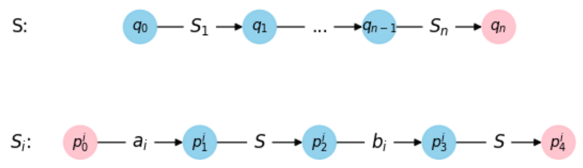

Figure 1: Possible RSM for Dyck language on n types of brackets. Red vertices $q_n, p_4^i\ \forall i \in \{1,\dots,n\}$ are final states, $q_0, p_0^i\ \forall i \in \{1,\dots,n\}$ are initial states in each box

COROLLARY 3.5. *If we fix the input grammar to grammar of Dyck language, then planarity of the Kronecker product is guaranteed if input graph is 1-contractible to a path or is a circuit.*

Necessary conditions for planarity of Kronecker product of digraphs. Let us consider the case of product of two planar digraphs. From [6] it is known, that Kronecker product of $K_{1,3}$ with itself if non-planar.

PROPOSITION 3.6. *If G_1 and G_2 contain subraphs $K_{1,3}$ oriented as in Fig. 2, 3, 4 or 5, then their Kronecker product is non-planar.*

□ Proof is a brute-force algorithm that checks planarity of Kronecker product for all possible orientations of pairs of graphs $K_{1,3}$.
If graphs G_1 and G_2 contain pair of subgraphs $K_{1,3}$ which Kronecker product is non-planar, then $G_1 \wedge G_2$ is non-planar too. ■

4 CONCLUSION

Conclusion and future work.
Efficient implementation?
!!!

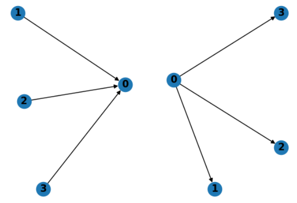

Figure 2: Orientation of the edges of $K_{1,3}$ graphs, which Kronecker product is non-planar: all edges of the graph at the right are oriented from vertex 0

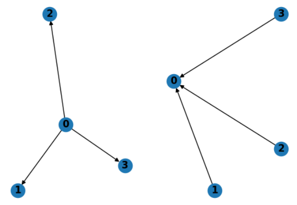

Figure 3: Orientation of the edges of $K_{1,3}$ graphs, which Kronecker product is non-planar: all edges of the right graph are oriented to vertex 0

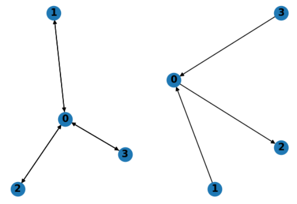

Figure 4: Orientation of the edges of $K_{1,3}$ graphs, which Kronecker product is non-planar: edges of the right graph are oriented 2 to vertex 0 and 1 from it

REFERENCES

[1] P. G. Bradford. 2017. Efficient exact paths for dyck and semi-dyck labeled path reachability (extended abstract). In *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*. 247–253. https://doi.org/10.1109/UEMCON.2017.8249039
[2] Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. 2019. Truly Subcubic Algorithms for Language Edit Distance and RNA Folding via Fast Bounded-Difference Min-Plus Product. *SIAM J. Comput.* 48, 2 (2019), 481–512. https://doi.org/10.1137/17M112720X arXiv:https://doi.org/10.1137/17M112720X
[3] Krishnendu Chatterjee, Bhavya Choudhary, and Andreas Pavlogiannis. 2017. Optimal Dyck Reachability for Data-Dependence and Alias Analysis. *Proc. ACM Program. Lang.* 2, POPL, Article 30 (Dec. 2017), 30 pages. https://doi.org/10.1145/3158118

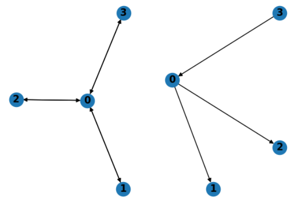

Figure 5: Orientation of the edges of $K_{1,3}$ graphs, which Kronecker product is non-planar: edges of the right graph are oriented 1 to vertex 0 and 2 from it

[4] Swarat Chaudhuri. 2008. Subcubic Algorithms for Recursive State Machines. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Francisco, California, USA) *(POPL '08)*. Association for Computing Machinery, New York, NY, USA, 159–169. https://doi.org/10.1145/1328438.1328460

[5] Krzysztof Diks and Piotr Sankowski. 2007. Dynamic Plane Transitive Closure. In *Proceedings of the 15th Annual European Conference on Algorithms* (Eilat, Israel) *(ESA'07)*. Springer-Verlag, Berlin, Heidelberg, 594–604.

[6] M Farzan and Derek A Waller. 1977. Kronecker products and local joins of graphs. *Canadian Journal of Mathematics* 29, 2 (1977), 255–269.

[7] Zvi Galil, Giuseppe F. Italiano, and Neil Sarnak. 1999. Fully Dynamic Planarity Testing with Applications. *J. ACM* 46, 1 (Jan. 1999), 28–91. https://doi.org/10.1145/300515.300517

[8] Ming-Yang Kao. 2008. *Encyclopedia of algorithms*. Springer Science & Business Media. 342–343 pages.

[9] Adam Karczmarz. 2018. *Data structures and dynamic algorithms for planar graphs*. Ph.D. Dissertation. PhD thesis, University of Warsaw.

[10] Lillian Lee. 2002. Fast Context-free Grammar Parsing Requires Fast Boolean Matrix Multiplication. *J. ACM* 49, 1 (Jan. 2002), 1–15. https://doi.org/10.1145/505241.505242

[11] Sairam Subramanian. 1993. A Fully Dynamic Data Structure for Reachability in Planar Digraphs. In *Proceedings of the First Annual European Symposium on Algorithms (ESA '93)*. Springer-Verlag, Berlin, Heidelberg, 372–383.

[12] Leslie G. Valiant. 1975. General Context-free Recognition in Less Than Cubic Time. *J. Comput. Syst. Sci.* 10, 2 (April 1975), 308–315. https://doi.org/10.1016/S0022-0000(75)80046-8

[13] Mihalis Yannakakis. 1990. Graph-theoretic methods in database theory. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. ACM, 230–242.

[14] Qirun Zhang, Xiao Xiao, Charles Zhang, Hao Yuan, and Zhendong Su. 2014. Efficient Subcubic Alias Analysis for C. *SIGPLAN Not.* 49, 10 (Oct. 2014), 829–845. https://doi.org/10.1145/2714064.2660213