

Pruning Algorithm on k-path vertex to Minimum the results

K.Saidi Reddy¹, B.Raja Rao²

^{1,2}Assistant Professor, Department of IT,

Malla Reddy Engineering College (Autonomous), Secunderabad, Telanagana

ABSTRACT

For a directed graph G with vertex set V we call a subset $C \subseteq V$ a k -(All-)Path Cover if C contains a node from any path consisting of k nodes. This paper considers the problem of constructing small k -Path Covers in the context of road networks with millions of nodes and edges. In many application scenarios the set C and its induced overlay graph constitute a very compact synopsis of G which is the basis for the currently fastest data structure for personalized shortest path queries, visually pleasing overlays of subsampled paths, and efficient reporting, retrieval and aggregation of associated data in spatial network databases. Apart from a theoretical investigation of the problem, we provide efficient algorithms that produce very small k -Path Covers for large real-world road networks (with a posteriori guarantees via instance-based lower bounds).

1. INTRODUCTION

The massive acquisition of geospatial data in the course of collaborative projects like OpenStreetMap (OSM) [1] or by companies like Google or TomTom has lead to a dramatic growth of data to be handled in Spatial Network Databases (SNDB). For example, the OSM 'world graph' at the beginning of 2007 contained less than 30 million nodes whereas in 2013 this number has grown to more than 2 billion nodes. A limit to this growth is nowhere to be seen due to the demand for a more and more accurate and detailed representation of our

environment. SNDBs manage geographic entities located in an underlying road network supporting efficient data retrieval operations, in particular taking into account connectivity properties of the road network. Google/Bing/ Yahoo Maps are all incarnations of SNDBs. Let us first look at a few applications for SNDBs which can benefit from small k -Path Covers.

Application 1: Assume we have computed or decided on some (not necessarily shortest or quickest) route γ for a weekend trip and are interested in shopping or refueling opportunities along the route. If we had a

data structure D which can retrieve for any node $v \in V$ in the network nearby points of interest like shopping malls or gas stations, we could query D with every node on π to get the desired answer. With π possibly consisting of hundreds or thousands of nodes, of course, it might be more efficient (but sufficiently accurate) to query D with only every k -th node on π (e.g., $k = 10$). Still, this requires having constructed D for all nodes of the network. An elegant solution approach is to identify a small subset $C \subseteq V$ such that for any path consisting of k nodes at least one node is contained in C and construct D for C only. Similarly, if the goal is to acquire statistics along paths (like the percentage of forest/desert coverage along routes throughout the USA), it is more efficient to aggregate that data at the nodes determined by C both in terms of space requirement as well as running time for a single query.

Application 2: In another scenario, a web portal dealing with scenic hiking trips sends several suggested hiking routes to the clients for visualization in a browser. In particular in a zoomed out view, it would be a waste of bandwidth to transmit every single node of each route across the internet; sub sampling the paths, e.g. every k -th node, is the preferred method. If routes overlap, though,

the chosen sub sampling for the routes should be consistent to avoid visual artifacts. Again, if we could determine a subsample C of V which guarantees that in any route at least every k -th node is present, a more pleasing visual appearance can be achieved when rendering an overlay of the subsample routes, see Figure 1.

Application 3: Next generation route planners allow for personalized route planning queries where each user has an individual profile depending on his personal preferences and driving styles. For example the typical driving speed varies (speeder or slow driver), people like to trade gas price against travel time, and different vehicles exhibit different turn costs and so on. So a query might not only consist of source and destination but also of a set of parameters which determine under which metric the optimal path in the road network is to be computed. If we construct an overlay graph on a small subsample C of V where C hits any k -path in the road network, we have a metric-independent compressed graph at hand. For any kind of input parameters specified by a user, we can compute the respective edge costs in this

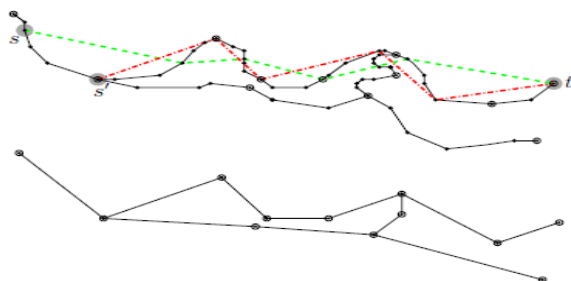


Figure 1: Small hiking map.

In the upper image the dashed red and green line indicate two hikingRoutes ($s \rightarrow t$ and $s_0 \rightarrow t$) simplified greedily using $k = 6$. While $s_0 \rightarrow t$ is a sub route of $s \rightarrow t$, the two simplified routes do not resemble each other at all. The circled nodes are a feasible global cover for $k = 6$, resulting in a nice simplification of the map presented in the lower image. Graph on demand and find the optimal graph on demand and find the optimal path by a search in The overlay graph considerably faster than a search in the original graph. We want to emphasize that for all these applications it is crucial that the path cover C hits any path consisting of k nodes, not only shortest or quickest paths under some fixed metric.

2. RELATED WORK

Tao et al. in [12] considered the problem of computing a k -Shortest-Path Cover, that is, a set of nodes $C \subseteq V$ such that C contains at least one node from every shortest path (under some fixed metric) consisting of k nodes. More concretely, they could, for

example, construct a set C which was only 15% of the size of V for $k = 16$ and the US road network. As one application example they showed how to use such a small C to accelerate shortest path queries (for a fixed metric) via the overlay graph induced by C (and in combination with additional speed-up techniques like reach [9]). We want to note, though, that the achievable query times have meanwhile been superseded by current speed-up techniques like Transit Nodes [3], Contraction Hierarchies (CH)

[8] or Hub Labels [2] which answer queries faster by several orders of magnitude. A fundamental restriction of all these techniques (including [12]) is that they all rely on fixed edge weights for the preprocessing stage. If edge weights change (e.g. for a different user profile), the preprocessing has to be revised or even redone from scratch again, making all these approaches unsuitable for the use-case where every query comes with a different user profile. In [6] a three phase approach was introduced to allow for customizable route planning (CRP). The first phase is a metric-independent graph preprocessing purely based on the topology of the graph. In the second phase some metric is considered and the graph is customized accordingly. In the third phase queries can

be answered efficiently for the metric specified in the second phase. But as soon as the metric changes phase two has to be redone. For the CRP framework, the metric customization takes several seconds (on a rather powerful multicourse machine). While this is tolerable for simulating traffic dependent edge costs or when customizing for a specific single user, some seconds are too slow when every query demands its own metric (in fact running plain Dijkstra might then be faster). We will take care of such highly dynamic queries by also applying a metric-independent preprocessing at first, but then merge the second and third phase such that metric changes due to different input parameters can be dealt with on query time. In [7] the authors adapt the CH preprocessing technique to the case where each edge e in the network has d associated costs $c_1(e); c_2(e); \dots; c_d(e)$. A query consists of source, destination and non-negative multipliers $\alpha_1; \alpha_2; \dots; \alpha_d$; the data structure returns the shortest paths under edge costs

$$c(e) = \alpha_1 c_1(e) + \alpha_2 c_2(e) + \dots + \alpha_d c_d(e).$$

Experimental results proved the technique to yield high speed-ups for $d = 2$ and 3 . Still, the speed-up for $d = 3$ was already considerably lower than for $d = 2$, as with each additional metric the hierarchy of

optimal paths (which is crucial for the approach) subsides. Moreover it was shown that considering metrics which are orthogonal to each other (like low travel time and preferring quiet roads at the same time) is disadvantageous.

For a larger number (e.g. $d > 4$) of unsimilar costs this approach comes at its limits.

3. OUR RESULTS

The main contributions of our paper are the following:

We generalize the work of [12] by considering all paths in the network instead of only shortest paths and devise efficient algorithms to compute small k -Path Covers. The resulting covers are even smaller than the covers reported in [12] but with the much stronger property of covering all k -paths instead of only shortest k -paths.

As a by-product we devise an algorithm that constructs considerably smaller sets C with the same properties (covering shortest k -paths) as the ones derived in [12]; for example, for the US road network and $k = 16$ we can find a set C which is only 5.8% the size of $|V|$ (compared to 15% reported in [12]).

More on the theoretical side we show that the problem of covering all k -shortest-paths can be approximated within a logarithmic factor of the optimal solution by using α -net

theory and proving a new result on the VC-dimension of directed shortest path systems. As a concrete application for our k-Path Covers we devise the to our knowledge to-date fastest scheme to answer personalized route planning queries.

Outline

After providing formal definitions in Section 2, we review in Section 3 theoretical results on the complexity of the k-Path Cover problem, and the k-Shortest-Path Cover problem respectively.

As k-Path Cover is NP-hard [4] and k-Shortest-Path Cover turns out to be NP-hard as well, we investigate approximation algorithms based on low VC-dimension of the underlying set system. In Section 4 we develop practical algorithms to construct k-All-Path Covers and k-Shortest-Path Covers as well as instance-based lower bounds. We conclude with an experimental section showing the practicability of our developed algorithms and in particular looking at the use case of personalized route planning.

4. CONSTRUCTING KPATH COVERS IN PRACTICE

In the following we will develop approaches for efficient cover set construction in practice. We will first discuss the general case of k-Path Covers but then also consider the special case of k-Shortest-Path Covers.

A naive approach that immediately comes to mind is to enumerate for each vertex $v \in V$ all paths with k nodes that start in v and store them. Then any (heuristic) Set-Cover/HittingSet algorithm, e.g. the greedy approach, could be used to retrieve a feasible cover C. Unfortunately, this is not practical for large input graphs, as the exploration time as well as the space consumption for extracting and storing all k-node paths is prohibitive. Therefore we will devise a more sophisticated approach, which allows for a considerably more efficient computation of a feasible cover C.

The Pruning Algorithm

We follow a pruning approach with the following high-level idea: Starting with all the nodes in the cover, i.e. $C = V$, we consider the nodes one by one, always deciding for a node v whether it is necessary to keep it in C to maintain the covering property. To decide whether a node v can be Pruned from C, we essentially have to make sure that there exists no k-node path over v which does not contain any other node from the current cover C. So we basically have to explore all outgoing and ingoing paths of v until reaching other nodes from C. If the combination of such an outgoing and ingoing path yields a concatenated path of length k, we have to keep v in C. A high-

level view on this procedure is given in Algorithm 1. To decide whether a node v can be pruned, the procedure

Algorithm 1 Procedure to decide whether node v is necessary

for k -APC-cover.

NodeNecessary(v, k)

Construct the set P_o of all outgoing paths from v not containing

any node in $C \setminus fvg$

if $\exists P_o$ with $|P_o| = k$ then

return true

end if

for all P_o do

search for the longest incoming path into v

not containing nodes in $(C \setminus \{v\}) \cap fvg$

if such path of length $k - |P_o| + 1$ exists then

return true

end if

end for

return false

Is called with nodeNecessary(v, k). It returns true if a k -node path exists which is only covered by v (that is, v cannot be pruned) or false if no such path exists (that is, v can be pruned). The procedure uses two subroutines enumerating all incoming and outgoing paths not containing a specific set of nodes $\{v\}$ these subroutines can be easily implemented very similar to depth first search (but with potentially exponential running time in k).

While these subroutines are naturally implemented in a recursive fashion (like depth first search), our implementation is stack-based which is much faster in practice due to the avoidance of the overhead of stack frame (de)allocation during the recursive calls. Also observe that both the construction of the set P_o as well as the search for an incoming path will only explore paths of length at most k since by assumption C was a valid cover before consideration of node v . And in particular at the very beginning, when C is almost the whole vertex set, these two steps abort almost immediately. This is one reason for the pruning approach to be much faster than the naive algorithm in practice. **Theorem 1.** The pruning algorithm produces a feasible and minimal k -Path Cover C . Proof. The pruning algorithm only discards a node v from C if all paths of k nodes containing v are covered by $C \setminus fvg$. Therefore throughout the algorithm we always maintain a feasible k -APC C . So especially after termination the resulting set C has to cover all k -paths. For minimalist, consider the moment when v is regarded but not pruned from C . In this case there was a path which contains v as the only node from the current C . Hence any node present in the final cover C has a witness path which

certifies its necessity for the cover. Therefore no node can be removed from the final C without invalidating the solution. Hence C has to be minimal in a set theoretic sense. Of course, minimalism of C does not imply that it is also minimum, i.e. of minimal cardinality amongst all possible covers. Observe that the order in which nodes is considered during the course of the algorithm highly incenses the solution quality (while not for feasibility or minimality). Intuitively, nodes with a low degree might not cover as many paths as nodes embedded in dense clusters. We will provide an experimental study measuring the incense of different node order schemes for pruning towards the end of the paper.

5. EXPERIMENTAL EVALUATION

Environment and Data Sets

Our C++ implementations were compiled using gcc 4.6.3 and benchmarked on a 3.2GHz intel i5-3470 with 16GB RAM. As benchmark data we used road networks extracted from the OpenStreetMap (OSM) project as well as the standard DIMACS road network graphs³, which were also used in [12]. Edge costs (in the single metric case) were set to travel times. Most experiments were conducted on the largest

graphs GER and USA, see Table 4.5 for an overview

of the characteristics of the used graphs. The values of d_{avg} , d_{max} and t_{dij} denote the average degree, the maximum degree and the time a random one to all run of Dijkstra's algorithm takes on average, respectively.

Constructing kAPC

Let us start with the pruning approach for constructing sets C covering all paths consisting of k nodes: we first examine how different node orders affect the quality and the running time of the cover construction. We considered the two largest networks GER and USA; fixing $k = 16$ we evaluated node orders both decreasing (-dec) as well as increasing (-inc) according to their node ID (id-) as given by the original graph $_le$, number of incoming plus outgoing edges (oi-), the order in which the recursive calls of a depth-first-search visit the nodes (dfs-), the order in which the recursive calls of a depth-first-search are completed on the nodes (comp-), and finally simply random order (rand). Intuitively it makes sense to prune away low-degree nodes first, and indeed pruning in increasing degree order (oi-inc) leads to much smaller cover sizes compared to oi-dec.dfs-dec and comp-inc tend to prune out nodes in dead-ends first which seems favorable to dfs-inc and comp-

dec. For the lower bounds the differences are not very pronounced, so throughout the following benchmarks we use the comp-inc order for both cover construction as well as lower bounds. Note that our instance-based lower bounds prove that our constructed covers are pretty close to optimal (at most a factor 1:7 larger for GER, a bit worse with a factor of 3:2 for USA) { in fact, they could be even closer to the optimum since the lower bound is probably not really tight. The construction times for the lower bounds are almost negligible.

6. CONCLUSION

We introduced the k-All-Path Cover optimization problem with the goal of computing compact yet faithful synopses of the vertex set of road networks. Our proposed pruning algorithm provides close to optimal results in practice and was experimentally proven to be very efficient on large graphs. For the special sub case of covering all k-node shortest paths as proposed by Tao et al. [12], we considerably improved their results in terms of running time and solution quality. But even for covering all paths consisting of k nodes, we could construct surprisingly small cover sets in reasonable time for moderate values of k. On that basis, we developed a completely new framework for answering personalized

route queries, where the user provides not only source and target but also weights for a given set of metrics. Our solution is based on a metric-independent overlay multigraph constructed upon our cover set in a preprocessing phase. While other route planners require a relatively expensive customization phase to adapt to personalized metrics, our approach allows to incorporate them in the overlay graph on the fly. This leads to a speed-up of an order of magnitude compared to Dijkstra's algorithm. While this already allows for real-time query answering, a natural direction for future research is to aim for query times in the order of milliseconds as achievable for fixed metric shortest path queries. We want to emphasize that our speed-up technique is somewhat orthogonal to speed-up techniques like A* and may be very well combined with them. This might be a good starting point to achieve

REFERENCES

- [1] I. Abraham, D. Delling, A. Fiat, A. V. Goldberg, and R. F. Werneck. Vc-dimension and shortest path algorithms. In ICALP, pages 690{699. Springer, 2011.
- [2] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road

networks. In SEA, pages 230{241. Springer, 2011.

[3] H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast routing in road networks using transit nodes. *Science*, 316(5824):566, 2007.

[4] B. Bre_sar, F. Kardo_s, J. Katreni_c, and G. Semani_sin. Minimum k-path vertex cover. *Discrete Applied Mathematics*, 159(12):1189{1195, 2011.

[5] H. Br•onnmann and M. T. Goodrich. Almost optimal set covers in _nitevc-dimension. *Discrete & Computational Geometry*, 14(1):463{479, 1995.

[6] D. Delling, A. V. Goldberg, T. Pajor, and R. F. F. Werneck. Customizable route planning. In P. M. Pardalos and S. Rebennack, editors, SEA, pages 376{387. Springer, 2011.

[7] S. Funke and S. Storandt. Polynomial-time construction of contraction hierarchies for multi-criteria objectives. In ALENEX, pages 41{54. SIAM, 2013.

[8] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388{404, 2012.

[9] R. J. Gutman. Reach-based routing: A new approach to shortest path algorithms

optimized for road networks. In ALENEX, pages 100{111, 2004.

[10] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. In SoCG, pages 61{71, New York, NY, USA, 1986. ACM.