DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-191

Approximate Subgraph Isomorphism for Image Localization

Vaishaal Shankar, Jordan Zhang, Jerry Chen, Christopher Dinh, Mattthew Clements, and Avideh Zakhor
University of California, Berkeley
Berkeley, CA 94720
{vaishaal,jordan.of.zhang, jc9100,chris.d,clements,avz}@berkeley.edu

Abstract

We propose a system for user-aided image localization in urban regions by exploiting the inherent graph like structure of urban streets, buildings and intersections. In this graph the nodes represent buildings, intersections and roads. The edges represent "logical links" such as two buildings being next to each other, or a building being on a road. We generate this graph automatically for large areas using publicly available road and building footprint data. To localize a query image, a user generates a similar graph manually by identifying the buildings, intersections and roads in the image. We then run a subgraph isomorphism algorithm to find candidate locations for the the query image. We evaluate our system on regions of multiple sizes ranging from $2km^2$ to $47km^2$ in the Amman,Jordan and Berkeley,CA,USA. We have found that in many cases we reduce the uncertainty in the query's location by as much as 90 percent.

1. Introduction

Image geo-localization, particularly in urban settings, is an important subject with applications in augmented reality, navigation, and surveillance. The ultimate goal of image geo-localization is to assign a latitude and longitude to an image. The most straightforward way to accomplish this is to use digital GPS and compass modules integrated into imaging devices to record pinpoint geographical data at the moment an image is taken. However, many useful images are not geo-localized at the instant they are taken, or are erroneously geo-localized. Multipath effects in cities with tall buildings is well-studied phenomenon known to have a large impact on GPS accuracy in urban regions.

Many approaches to image geo-localization focus on reducing the set of possible candidate locations for the image. [19] and [1] have had success with localization for desert or mountain images, by matching skylines to a database of model skylines computed from a digital elevation map. In some cases the area of interest is reduced by as much as 99%. Ramalingam et. al. in [15] apply a similar method on urban skylines to address the inaccuracy of GPS in urban canyons, but their method requires images taken directly upward that encompass a full 360° spherical dome above the camera location, a condition that few street level images meet.

Much previous work in high precision urban localization relies on distinctive landmarks and on matching image features, such as SIFT, with a database of similar features gathered from sources such as Google Street View [16]. Methods such as [9], [11] and [8] use large geotagged image databases to solve this problem on a global scale. If a readily available source such Street View has low coverage in an area of interest, then a working database for feature matching methods is nearly impossible to generate. In a proof of feasibility, Bansal et. al. [2] claim that overhead imagery is essential to overcoming the sparse coverage of ground image databases by demonstrating a rough facade matcher for oblique satellite images. Even if image matching localization methods succeed, they result in large candidate areas, often the size of a city or country. The authors of [7], for example, forgo pinpoint localization entirely and only determine whether a picture is from Prague or Paris. Furthermore image features such as SIFT, while effective for general image retrieval and identification, fail to capture important spatial features for image localization.

In this paper, we exploit the inherent structure of urban environments, the fact the buildings are constructed in an orderly fashion along streets with geometrically regular intersections to localize images. Identifying the existence and type of the roads, intersections and buildings in the image can provide rich, invaluable information to aid localization. There are readily available sources, such as Open Street Map (OSM), for road and building footprint data world wide. Numerous satellite based building detection algorithms and road detection algorithms such as [14] can be used to generate road and building footprint data. We take advantage of the orderly layout of cities to generate a database graph representing the roads and buildings in an area of interest. To localize an image, a human user constructs a similar graph by identifying all buildings, in-

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-191

tersections, and roads in a query image. Our system then takes in these two graphs and runs a custom made subgraph isomorphism algorithm inspired by [12] to find candidate subgraphs in the georeferenced database graph. Our graph matching algorithm prunes candidate subgraphs using shadow analysis, [4], and geometric consistency techniques. The candidate subgraphs are georeferenced so the location of the nodes in subgraphs corresponds to potential locations for the query graph.

This paper is organized as follows: Section 2 reviews related work in graph matching. In Section 3 we describe the semantics of the graph, and design decisions in choosing this particular graph model. In Section 4 we discuss our graph generation method for the area of interest and query image. We then outline the details of our graph matching algorithm in Section 5. In Section 6 we detail our experimental results. Finally in Section 7 we conclude with a summary of our contributions and future work.

2. Related Work

The main component of the proposed image geolocalization method is computing the graph matches, also commonly known as graph isomorphisms or homomorphisms [5]. Section 2.2 of [3] and Sections 1 and 2 of [5] provide overviews of the commonly studied forms of graph matching. Of the graph-based image geo-localization papers in the literature, many use graphs as a tool to assist feature matching to a ground level database. For example, [21] uses a minimum clique algorithm to match pruned SIFT features from a ground level database.

Among the various graph matching problems, inexact attributed subgraph matching holds the most relevance. In this formulation, we are given two graphs (V, E) and (V', E') such that $|V| \geq |V'|$, and the goal is to find a mapping $f : V' \rightarrow V$ such that (V', E') is isomorphic to a subgraph of (V, E). Since both the building and road databases and the user-generated query input are error-prone, our subgraph isomorphism algorithm must be robust to uncertainties in both the database and query graphs. Furthermore, our graphs use attributes, metadata on the nodes and edges themselves, to facilitate the match. Finally, to accommodate large regions of interest in which a particular query image might have been captured, the graph matcher must scale to large data sets.

The bulk of subgraph matching literature is concerned with subgraph matching in which neither query nor database graphs have noise[6, 17, 20]. Many error-tolerant graph matchers such as [13], [10], [22] are designed to compute similarity metrics on the entirety of *small* graphs rather than finding a small piece of a large graph that matches a query graph, and are not suitable for our purpose. The matcher proposed in [18] scales well, handles attributes on the graphs, and returns best-effort matches in the presence of errors in the graph. However, its robustness to common errors in our geolocalization problem, such as mislabeled building colors or heights, spurious edges, and omitted buildings as a result of occlusion in the query image, is limited. The algorithm proposed by [12], tolerates excessive errror on the database side by treating the entire database graph as Markov random field, and attempts to find high probability matches. This algorithm scales well and finds accurate matches on large graphs but it is not tolerant of errors on the query graph.

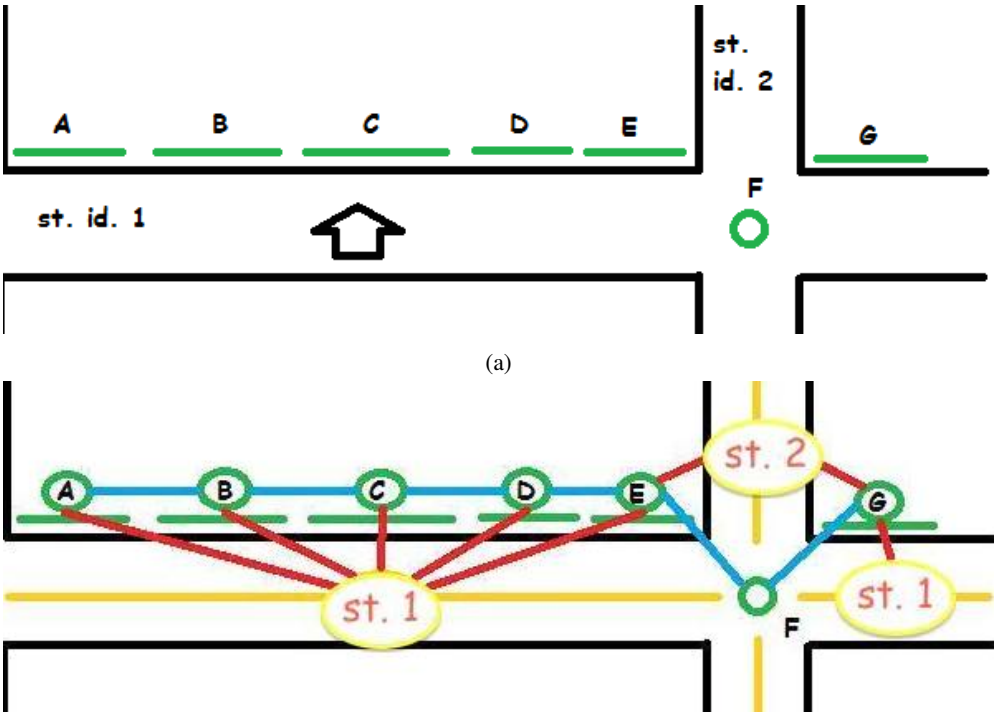
3. Graph Representation

As seen in Figure 1b The nodes in our graph represent real world entities that are easily recognizable from a query image by humans: buildings, intersections, and streets. The edges in our graph represent "logical links" that exist between these real world entities, and are easily recognizable from the query image by a human. An edge between two buildings denotes that the two buildings are adjacent, while an edge between a building and a road means that the building sits on that road. In our graph, the parity of relationships between building and intersections is preserved. For localization to work properly, these edges must be easy to infer both from a ground level query image, and a map of building footprints. For the remainder of the paper we refer to the graph generated from the map of building foot prints as the *database graph* and the graph generated from query image as the *query graph*. The database graph's nodes are georefenced, while the query's are not.

Our graph contains annotations for the nodes, allowing easy differentiation between "building", "road" and "intersection" nodes. We also use these annotations to embed other information into the nodes. For example, for every intersection we also store its *degree*, the number of street blocks going into that intersection. For every road we store its *classification*, whether its highway, residential, arterial road, and its approximate *heading*. While we solely use these features for this paper, our system is flexible and can be easily expanded to include additional features. Figure 1a shows a sample set of buildings and an intersection. We generate one building node for each A,B,C,D,E,G, an intersection node for F, and two road nodes for streets 1 and 2. All the buildings have edges to their adjacent buildings, as well as to the road 1 node. Additionally E and G have an edge to the street 2 node, and the intersection F node. Figure 1b shows all the nodes and graph in the example enumerated.

Since all the nodes and edges in our graph are easily identifiable in a query image, we can create a similar graph for any query image with interaction from the user. We show an example query graph superimposed onto its query image in Figure 2a. The buildings are denoted by red circles, the intersections by blue circles, and the road by a gray

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-191



(a)

(b)

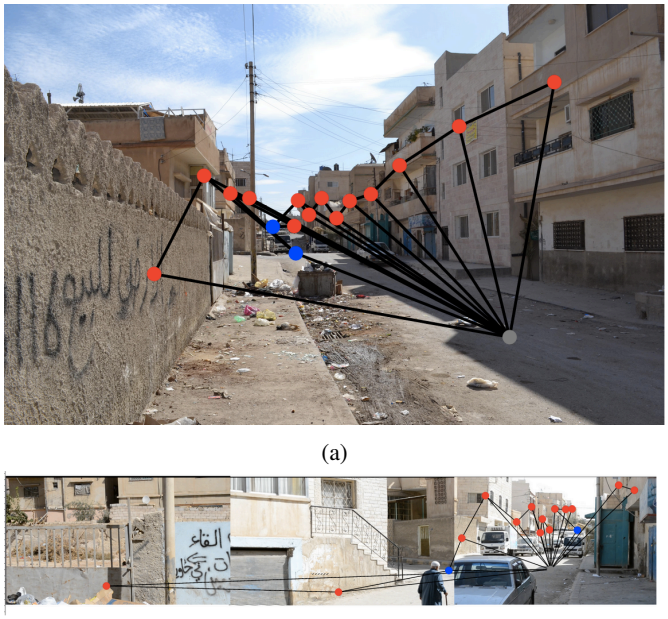
Figure 1: (a) Example buildings and intersections; (b) graph superimposed on the buildings and intersections.

circle. This graph model is anchored to real world georeferenced entities, so it is independent of features of the actual query image such as camera field of view, time of day and lighting. Nevertheless there is still room for measurement error from the human user. For instance, certain queries have occluded intersections that ultimately make the query graph look different from the correct graph representation of the same location in the database graph. We account for this uncertainty by adding edge weight between 0 and 1, corresponding to uncertainty. We then connect non adjacent buildings such as those across an intersection with "low weight" edges. The value of these low weights, and how aggressively to add these edges can be a tuneable parameter. These weights are used in the matching algorithm to be covered in Section 6.

4. Building Database Graph

Even a small town has thousands of buildings and intersections, so it is imperative to automatically generate the database graph for a geographic region. Fortunately, urban environments have a great deal of easily exploitable structures and patterns when it comes to building the graph.

We begin with a set of building footprints and OSM roads for the area of interest. The building footprints are either generated manually or obtained from OSM. We then generate a set of intersections from the roads. The set of buildings centers, roads, and intersections constituite all the nodes in our graph.



(a)

(b)

Figure 2: Example graphs for (a) an image, (b) video query.

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-191

Before adding any edges, we first preprocess to generate KD-trees for building centroids and road segments midpoints. This improves the performance of our algorithm since it relies heavily on nearest-neighbor search in 2-D space. We begin by adding edges between intersection nodes and the road nodes that make up the intersection. This step is a rather simple as the edges are explicitly given by the OSM data.

Next, we link buildings on a road with a node representing that road. This process is shown in Figure 3. For each building b_i we find its K_r closest roads using our precomputed road segment KD-tree and call this set S_i. We also query our building KD-tree to find its K_b closest buildings and call this set B_i. For each road S_{i_k} in S_i, we then extend a vector v_i from b_i to S_{i_k}. Then, for every neighbor building in B_i and every neighbor road in S_i that is not S_{i_k}, we check whether any of them intersects v_i. If not, we draw an edge from b_i to S_{i_k}. This algorithm preserves the semantic meaning for a building to be "on a road", since its "view" of the road is unobstructed by any other building, and it is "close" to the road.

Having generated edges from buildings to the road(s) they belong to, we connect buildings to their neighbors. We precompute a KD-tree consisting of the centroids of the building footprints. Then for each building b_i we enumerate its K_b nearest neighbors denoted by N_i. Let R_i be the set of roads that b_i has an edge to. Then for each building b_k in N_i we discard the ones where $R_i \cap R_k = \varnothing$ i.e when b_i and b_k share no roads. Then we filter the remaining candidates and remove any b_k where the shortest path from b_i to b_k crosses a road. By definition, adjacent buildings cannot sit on opposite sides of any roads. Last, we connect b_i with its nearest neighbor b_k. Finally, to connect intersections to their adjacent buildings, we simply take the intersection x_i and connect it to its nearest neighbor building on each block the intersection is connected to. We can modify this process to add the previously mentioned "low-weight" edges to the next K_n nearest neighbors in both the intersection and building case.

We opt to have humans manually generate the graphs for queries as shown in Figure 2. Specifically our system has a web user interface that allows for a user to upload a query image and click around the image to generate all the necessary nodes and edges. For video queries we stitch together frames to cover the entire field of view of the camera as shown in Figure 2b. The output format for this query graph is identical to the format of the database graph. An example of the graph for an image with one road, two intersections, and 14 buildings is shown in Figure 2b.

5. Subgraph Isomorphism Algorithm

The main component of our method is the approximate subgraph isomorphism algorithm, which allows for efficient retrieval between the query graph and the large georeferenced database graph. Subgraph isomorphism is known to NP-Hard [3]. It is a well studied problem with many heuristically optimized and specialized algorithms that perform well in practice. We have based our algorithm on the error tolerant path matching approach by Moustafa *et al* [12]. A path is defined to be a collection of nodes and edges. A example of a path in Figure 1b is $[A, B, C, D, E]$.The path matching approach searches for matches of individual paths in the subgraph to find a consistent subgraph within these path matches. While [12] finds isomorphisms between two feature-rich graphs with the same number of features on both the query and database side, our algorithm utilizes the position and geometric information found in the database not present in the query graph to further prune the search space.

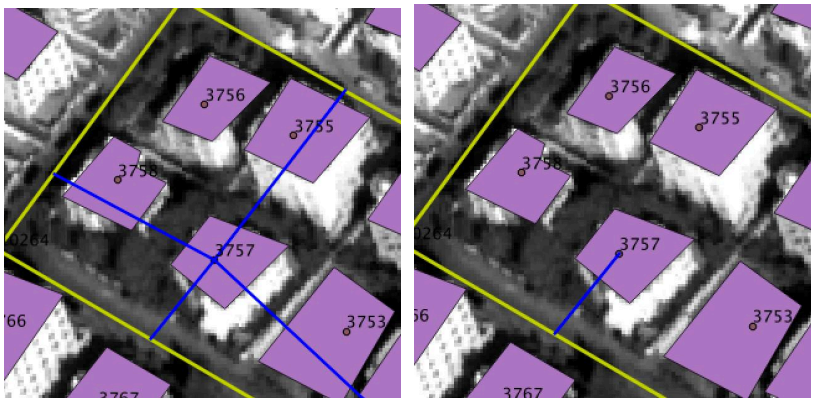
Furthermore due to the geometric nature of our problem, our nodes have a low maximum degree resulting in a lower algorithmic complexity than the one in [12]. The number of paths in a regular graph without geometric constraints is $O(N^N)$ whereas the number of paths in our geometrically constrainted graph is $O(k^N)$ where k is a small constant corresponding to the maximum degree of any node in our graph, and N is the total number of nodes in our graph, which could be a very large number. Empirically we have found k to be roughly 7 in our datasets.

Our system consists of an offline and online phase as shown in Figure 4. The offline phase runs once on the database graph and stores the graph and certain precomputations in an efficient manner. The goal of the offline phase is to generate an efficient lookup structure for query paths. We generate this structure by decomposing the database graph into all paths potentially visible by query images and "summarizing" each path by the information visible at query time. The online phase also decomposes the query graph provided by the user into a set of paths. Then it uses the previously generated lookup structure to efficiently find a list of candidate paths for each query path, which are then pruned using a variety of techniques to achieve a small set of final candidate subgraphs. Next, we explain each phase in more detail.

5.1. Offline Database Generation

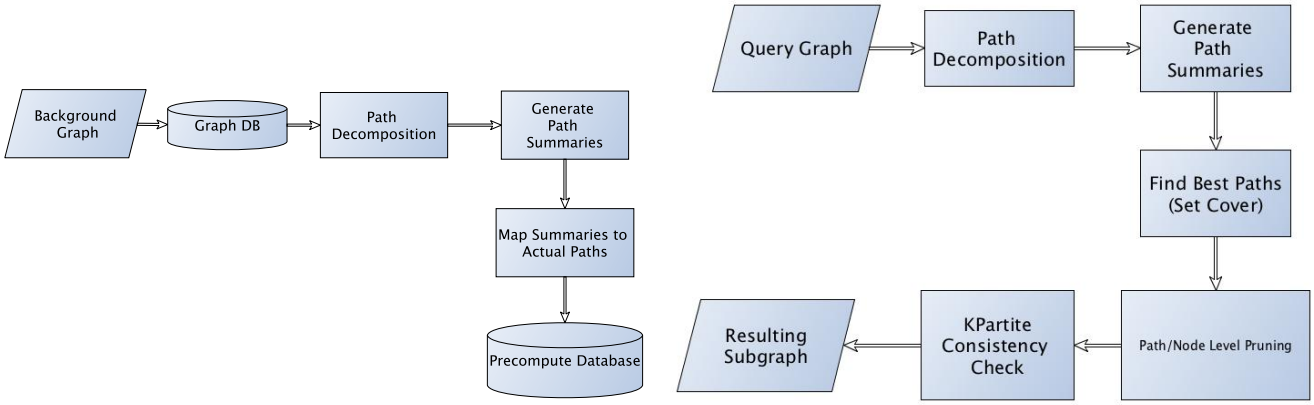
The offline phase consists of 3 key steps as outlined in Figure 4. The first and arguably most expensive part of the offline phase is the path decomposition. One possibility is to extract every path of length less than N_p, an input parameter of our algorithm. However extracting all the paths would yield poor performance. As such, it is desirable to only extract paths visible in a query image. For instance, paths that make sharp turns and span multiple orthogonal roads are highly unlikely to be observed in a query. Thus, intuitively a path consists of a row of adjacent buildings on

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-191



(a) (b)

Figure 3: The two steps involved in attaching a road with a building. First in (a) we draw a vector from the building to all the nearby roads. Then in (b) we only keep the vectors that do not cross other buildings.



(a) (b)

Figure 4: (a) Offline phase block diagram (b) Online phase block diagram.

one road excluding the building on an intersection.

Formally, for each path p_i let N_i be the set of neighboring roads of the buildings on that path [1], r_k be roads in the database, and b_k buildings in the database. That is

$$N_i = \{r_i : \exists (r_k, b_k) \in edges, b_k \in p_i, b_k \neq intersection\}$$

If N_i is not a singelton, we discard p_i as a path reflecting the fact that all buildings on a path are one road. For each path in our final path decomposition, we multiply all the edge weights along the path to obtain a *path confidence* value pc_i stored along with the path. An example of a valid path in Figure 1b is $[A, B, C, D, E]$.

[1] In computing N we of course ignore corner buildings in the path as they are always on two roads.

After extracting all "interesting" paths from the database graph, we generate a "path summary". These path summaries, consist of low error, low entropy attributes for the nodes in the path, and are reliably consistent between the query and database. For example the path $[C, D, E, F, G]$ from Figure 1b would be summarized as $[(bld), (bld), (bld), (deg4int), (bld)]$, where *bld* stands for building and *deg4int* stands for a four way intersection. We can add more features to this step to make it more discriminative at the cost of increasing the number of unique summaries.

Next for each unique summary S_i, we take the list $(PL)_i$ of all paths whose summaries are S_i and insert the key value pair $(S_i, (PL)_i)$ into an persistent key value store. We also store a histogram H of our summaries to keep track of the number of paths that are summarized to S_i. For each S_i

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-191

we store the number of unique paths in the list $(PL)_i$. as $H[S_i]$, i.e

$$H[S_i] = |(PL)_i|$$

We use this quantity later in Section 5.2 for choosing a reasonable path decomposition of the query graph.

5.2. Online Query Localization

The online phase of our system consists of 5 key steps as outlined in Figure 4b. The first three steps break down the query graph into paths, summarize the paths, and use the previously generated path summary key value store to find candidate paths in the database graph for each query path. The final two stages use attributes stored in the nodes and geometric constraints imposed by the query image to prune the number of candidate paths.

Path decomposition in the query side works almost identically to its offline counterpart, the only advantage being that the query graph is quite small, and its path decomposition can be computed rapidly. In order to find a subgraph match we need to match every node in the subgraph to a candidate node in the database graph. To do this on a path by path basis, we need a set of paths that "cover" the entire query. In other words the union of the nodes and edges in all the paths chosen, must equal all the nodes and edges in the query graph.

Not all path covers are equally desirable. Certain paths are highly discriminative because their path summary maps to fewer unique paths. These usually yield faster and better localizations. We model the path cover problem as an instance of set cover. Rather than minimizing the total number of sets, we minimize the total $cost$. This cost for each query path P_i is defined to be the histogram value for the summary S_i computed for the path, $H[S_i]$ divided by the number of nodes in P_i. Intuitively this cost assigns "rarer" and longer paths a lower cost, since a smaller histogram value means the path is discrimintive, and a longer path means it has more information. Set cover is also NP complete, but for this particular instance, the standard greedy approximation suffices. As an example the cover set for Figure 2a would be a path consisting of all buildings on the right side of the road, and a path consisting of all the buildings on the left side and the two intersections.

After solving the instance of set cover, we have a set of the most discriminative paths P containing all the nodes and edges in the query. When computing the cost we ignore any paths whose confidence is lower than α, a runtime parameter of the matcher. We then generate a mapping from each query path to its matching candidate paths from the database.

Next we wish to reduce the number of candidate paths. This is done in 3 steps, node level, path level, and geometry level. For node level pruning, we enumerate all the nodes in all candidate paths. Each node n_i in the candidate node list with a matching query node q_k must have neighbors consistent with q_k. The neighbors of n_i is defined to be the all nodes that have an edge to n_i. Since the query is a subgraph, at no point can a matching node in the database graph have fewer neighbors of any type than a query node. Node level pruning then prunes any nodes that do not meet this criterion and their associated paths. We can also potentially prune other node level characteristics not included in our path summary here, such as height, color or footprint area.

Next, we examine *path level characteristics*, namely road type, and path direction. OSM provides road classifications for every road in our area of interest, and the user can approximately classify a road in the query, since its quite easy for a human to differentiate between a highway and a residential road. Using these classifications, we discard candidate paths that are on inconsistent roads with respect to the query.

Using shadow analysis techniques from [4], the user can approximate the heading of the roads in the query image. This combined with the existing headings for roads provided by OSM allow us to eliminate more inconsistent paths. In practice we have found this method to be effective in discriminating between north-south and east-west roads.

Finally we enforce the geometry between the paths in the query to be preserved. We look for three geometric relationships:

1. Two paths sharing a road with each other.
2. Two paths being geographically"close" to each other.
3. Two paths overlapping.

To deal with all of the above at once we generate a *KPartite Graph*. This graph is a metagraph where there is a node for each candidate database path and a node for query path. We draw an edge between two nodes in this graph if they share one of the above relationships. We then use the edges to enforce candidate paths to have the same geometric relationships the query paths have. These edges have a label e_k denoting the types of the relationship it signifies. Query path nodes only have edges to other query path nodes, and candidate path nodes only have edges to other candidate path nodes.

For every two query paths q_i and q_j that share a particular edge relationship e_k, we enforce every candidate path of q_i to have an edge of type e_k to a candidate path of q_j. A candidate path without such an edge is geometrically inconsistent and is removed from our candidate list. We then iterate this algorithm to a fixed point, when no paths a removed

The above algorithm is invariant to the types of e_k. Currently our system supports the three geometrical consistency points listed above, but can easily be expanded to sup-

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-191

Query #	Query Size (Nodes, Edges)	# Query Intersec-tions	Search Space Reduction
002	17,31	2	53.1
012	16,29	1	23.9
107	12,21	2	170
142	18,33	2	65
192	4,5	1	6.4
197	15,28	1	57.5
B1	6,9	1	9.6
B2	7,11	1	50.5
B3	9,15	1	62.4
B4	11,19	1	8.2
B5	12,21	2	50.5
B6	12,21	2	161.8
B7	7,11	0	2.14
B9	9,15	1	8.2
B10	8,13	1	50.5
B11	7,11	1	53.4
B12	7,11	1	7.4
B13	6,9	1	11.6
B14	9,15	1	8.2
B15	6,9	1	8.4
B16	9,15	1	12.0
B17	10,17	1	12.0
B18	9,15	1	12.0

Table 1: Performance Analysis for 6 queries in Amman, Jordan and 18 queries in Berkeley, CA.

port more since the pruning algorithm remains unaltered. For every pair of database paths (p_i, p_k) we draw an edge of type "close" between them, if their Euclidean distance is less than d. This parameter is set by the user depending on the extent of the query. All query paths are considered "close" to each other. We draw an edge of type "road" if (p_i, p_k) are on the same road. This relationship only exists between two query paths if they share a road. Finally we draw an edge of type "overlap" if (p_i, p_k) overlap at any point. Again this relationship is mirrored on the query side. In practice we have found this KPartite matching to be highly effective in pruning the search space for our localization.

Finally after all the pruning, we connect the nodes in our resultant paths to get a subgraph that represents all the locations the query graph could exist in.

6. Evaluation and Results

We use the graph database engine Neo4j to store the database graph, and use MongoDB to store our path precomputations. We have tested our algorithm on queries

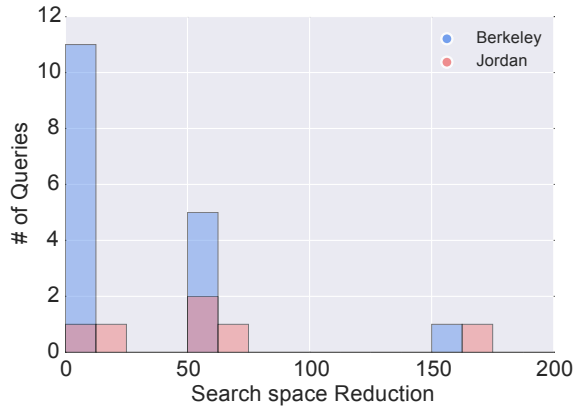


Figure 5: Search space reduction histogram.

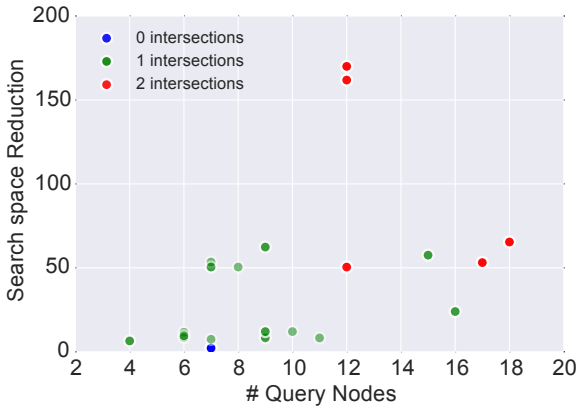


Figure 6: Search space reduction as a function of the size of the query graphs.

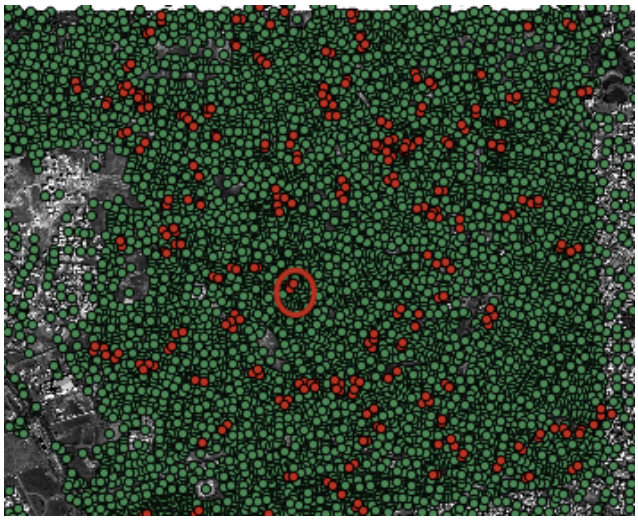


Figure 7: Example localization for query #197; red represents location of candidate subgraphs, green is all possible locations in the area of interest, and ground truth is circled.

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-191

in various locations in Amman, Jordan and the city of Berkeley, CA. We manually generated a building footprint database for Amman and its suburbs and used OSM footprint database for Berkeley. The ground truth location of the queries are known. Data for roads and intersections in both locations is from OSM.

Since our primary motivation is user-aided geolocation, typical metrics such as the percentage of top-1 matches are ill suited for evaluating our system. Rather, we opt for a metric involving the number of subgraphs left in our area of interest after a query has been provided. Each one of these subgraphs represents one particular localization of our query. We define search space reduction to be

$$\frac{\text{\# nodes in Area of Interest}}{\text{\# candidate subgraphs}}$$

The three evaluation regions in Jordan are of size 27km2 for query 197 with a graph size of $(10300; 14500)$, $3km^2$ for queries 002, 012, 107 and 142 with a graph size of $(3400; 5200)$ and $13km^2$ for queries 107 and 192 with a graph size of $(3400; 5200)$ where the first and second numbers in graph size correspond to the number of notes and edges respectively. The evaluation region in Berkeley is of size $47km^2$ with graph size of $(28000; 27000)$. We use a maximum path size of 10 for database generation.

Table 1 summarizes the results for 6 queries in Amman and 18 queries in Berkeley. In the best case corresponding to a query with two intersections we reduce the search space by a factor of 170, and in the worst case corresponding to a query with no intersections, the reduction is a factor of 2. The histogram of search space reduction is shown in Figure 5. As seen the median search space reduction is 12, and the mean is 39.4. In Figure 6 we plot search space reduction as a function of number of query nodes and color the points to indicate the number of intersections present in the query. As seen, intersections are critical in geolocating images. Figure 6 shows the best performing queries have 2 intersections in the field of view, while the worst performing query has no intersections. This is expected as intersections, though prevalent in an urban landscape, are rare enough that a pattern of buildings between two intersections forms a distinctive signature. We also observe in Figure 6 an inverse correlation between query graph size and resultant subgraph size. As expected, larger graphs are more informative of the scene, and should yield better localization. Figure 7 shows the resultant subgraph for query 197 in Jordan. The green dots denote all the nodes in our region of interest graph, the red dots represent the locations of the resultant subgraphs, and the ground truth is circled. As seen the candidate subgraphs are more or less uniformly spread throughout the region of interest, but the area they take up is a small fraction of the entire area. When there are multiple intersections of varying degree, the system performs well, as these patterns are "rare". When these conditions are met, we obtain as few as 20 resultant subgraphs or reduction factor of 170, for most queries entire pipeline runs in less than 360 seconds.

We now examine detailed statistics of query # 142 in Table 1 shown in Figure 2a as it is processed through our system. The area of interest for this query has 3,400 nodes and 5,200 edges. Database generation created 107,331 unique paths, and 178 unique path summaries. We ran the online phase with an α value of 0.5 and it decomposed the query graph into 6 covering paths. Initially there were 340 candidate subgraphs and after node, shadow, and path pruning there were 200 candidates remaining. The geometry pruning finally brought the candidate count to 64. The entire pipeline for this query takes 300 seconds, where the database generation took 31 seconds, path and node level pruning took 15 seconds, and geometry pruning took 224 seconds. Though geometry pruning took the most time, it is by far the most effective in reducing the number of candidates.

7. Conclusions and Future work

In this paper, we have presented a system for user aided image localization by exploiting the inherent graph like structure of urban streets, buildings and intersections. We reduce the search space in an area of interest by solving an approximate subgraph isomorphism problem in an efficient manner. Experimental results for Berkeley, CA and Amman Jordan showed search space reduction between 2 and 170, depending on the number of intersections.

By far the largest shortcoming of the system is lack of features. Currently our node attributes consist of node type, intersection degree and road type. There are many easily observable characteristics for buildings, intersections and roads, such as color, height, road elevation, and presence of vegetation that we could easily exploit to significantly prune our search space. Other areas of future work include automatic detection of buildings and roads from satellite images for areas with no OSM data, and automatic construction of the query graph.

8. Acknowledgments

Supported by the Intelligence Advanced Research Projects Activity (IARPA) via Air Force Research Laboratory, contract FA8650-12-C-7211. The U.S.Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, AFRL, or the U.S. Government.

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-191

References

[1] G. Baatz, O. Saurer, K. Köser, and M. Pollefeys. Large scale visual geo-localization of images in mountainous terrain. In *Computer Vision–ECCV 2012*, pages 517–530. Springer, 2012. 1

[2] M. Bansal, H. S. Sawhney, H. Cheng, and K. Daniilidis. Geo-localization of street views with aerial image databases. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 1125–1128. ACM, 2011. 1

[3] E. Bengoetxea. Inexact graph matching using estimation of distribution algorithms. *Ecole Nationale Supérieure des Télécommunications, Paris*, 2002. 2, 4

[4] M. Clements and A. Zakhor. Shadow analysis for camera heading in image geo-localization. 2, 6

[5] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298, 2004. 2

[6] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub) graph isomorphism algorithm for matching large graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(10):1367–1372, 2004. 2

[7] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. A. Efros. What makes paris look like paris? *ACM Trans. Graph.*, 31(4):101, 2012. 1

[8] A. Gallagher, D. Joshi, J. Yu, and J. Luo. Geo-location inference from image content and user tags. In *Computer Vision and Pattern Recognition Workshops, 2009. CVPR Workshops 2009. IEEE Computer Society Conference on*, pages 55–62. IEEE, 2009. 1

[9] J. Hays and A. A. Efros. Im2gps: estimating geographic information from a single image. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 1

[10] H. Hu, G. Li, and J. Feng. Fast similar subgraph search with maximum common connected subgraph constraints. In *Big Data (BigData Congress), 2013 IEEE International Congress on*, pages 181–188. IEEE, 2013. 2

[11] T.-Y. Lin, S. Belongie, and J. Hays. Cross-view image geolocalization. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 891–898. IEEE, 2013. 1

[12] W. E. Moustafa, A. Kimmig, A. Deshpande, and L. Getoor. Subgraph pattern matching over uncertain graphs with identity linkage uncertainty. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 904–915. IEEE, 2014. 2, 4

[13] M. Neuhaus and H. Bunke. An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 180–189. Springer, 2004. 2

[14] R. Peteri, J. Celle, and T. Ranchin. Detection and extraction of road networks from high resolution satellite images. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 1, pages I–301. IEEE, 2003. 1

[15] S. Ramalingam, S. Bouaziz, P. Sturm, and M. Brand. Skyline2gps: Localization in urban canyons using omni-skylines. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3816–3823. IEEE, 2010. 1

[16] G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–7. IEEE, 2007. 1

[17] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. *Proceedings of the VLDB Endowment*, 5(9):788–799, 2012. 2

[18] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 737–746. ACM, 2007. 2

[19] E. Tzeng, A. Zhai, M. Clements, R. Townshend, and A. Zakhor. User-driven geolocation of untagged desert imagery using digital elevation models. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*, pages 237–244. IEEE, 2013. 1

[20] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976. 2

[21] A. Zamir and M. Shah. Image geo-localization based on multiple nearest neighbor feature matching using generalized graphs. 2014. 2

[22] Q. Zhang, X. Song, X. Shao, H. Zhao, and R. Shibasaki. Attributed graph mining and matching: An attempt to define and extract soft attributed patterns. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1394–1401. IEEE, 2014. 2