

# GOOGLE HACKS™

**2nd  
Edition**  
Covers Gmail

*Tips & Tools for Smarter Searching*



***Tara Calishain & Rael Dornfest***

*With a new foreword by  
Craig Silverstein, Director of Technology, Google*

**O'REILLY®**

HACK  
#52

## Google Cartography: Street Art in Your Neighborhood

car-tog-ra-phy n. The art or technique of making maps or charts.

Google Cartography (found here: <http://richard.jones.name/google-hacks/google-cartography/google-cartography.html>) uses Google via the [Google Search API](#) [Chapter 9] to build a visual representation of the interconnectivity of streets in an area.

This application takes a starting street and finds streets that intersect with it. Traversing the streets in a breadth-first manner, the application discovers more and more intersections, eventually producing a graph that shows the interconnectivity of streets flowing from the starting street.

Figures 3-5 and 3-6 show maps generated for two of the world's great cities, New York and Melbourne, respectively.

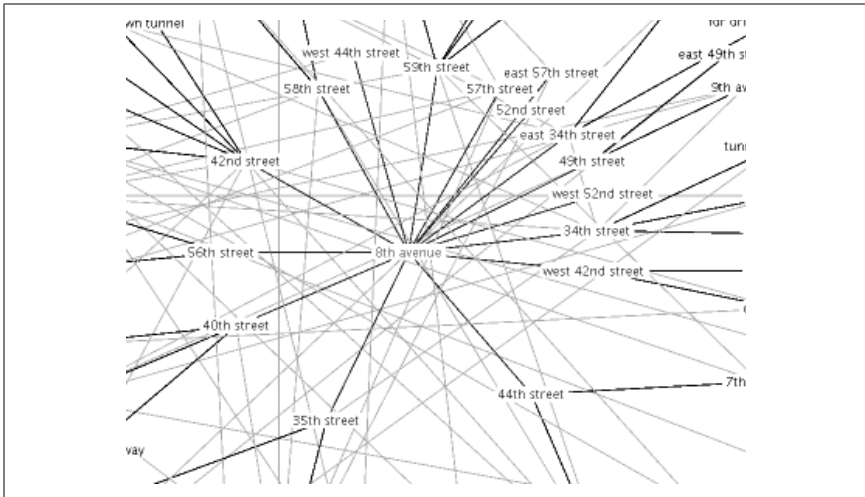


Figure 3-5. New York, U.S., as determined by Google cartography



If you know the streets in the areas shown, you will be able to find inconsistencies introduced by the text parsing process, explained in more detail in the following section.

### The Gory Details

Google Cartography uses the Google API to find web pages that refers to street names. Initial street and region criteria are combined to form a search query, which is then executed by the Google API. Each URL from the

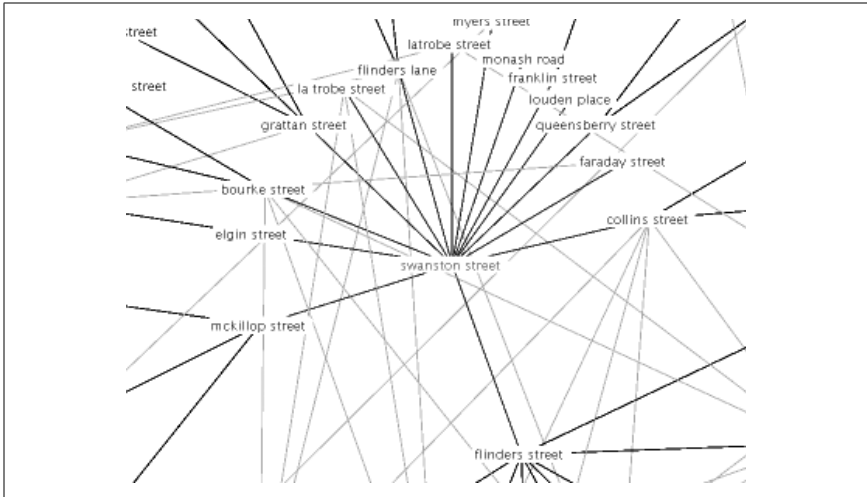


Figure 3-6. Melbourne, Australia, mapped by a little Google cartography

Google results is fetched and the content of the pages converted into text. The text is then processed using pattern matching (programmers, read: regular expressions) designed to capture information relating to the relationship between streets (for example, streets that cross each other or turn into other streets).

For each page a list of vertices is produced, where each vertex represents an intersection between two streets. After the results for the initial street have been processed, the list of vertices will hopefully contain some vertices that intersect with the initial street.

At this point, the mapper performs a breadth-first search through the streets that can be reached from the initial street. Each street traversed during this process is subjected to the same process as the initial street, expanding the list of known street intersections. This process continues until no more streets can be reached from the initial street or until the halting criteria is satisfied (for instance, reaching the maximum amount of Google API key usage).

Once the data collection phase has completed, the application converts the intersection vertices into a graph. Each street becomes a vertex of its own, with outgoing edges connecting to the vertices of intersecting streets. The connectivity of this graph is analyzed (using the Jung graph package at <http://jung.sourceforge.net>) to determine the largest maximal subgraph where all pairs of vertices are reachable from each other. This subgraph almost always contains the starting street; the probability of this not occurring should

decrease proportionally to the number of streets traversed (which naturally selects for streets that are in the same subgraph as the starting street).

The largest connected subgraph is then visualized using a Radial Layout algorithm provided by the Prefuse graph visualization framework (<http://prefuse.sourceforge.net>). The graph is initially centered on the start street but will automatically adjust its focus to center around the most recently selected street.

Ignoring its general lack of usefulness (at the time of this writing), there are several problems with the application worth noting:

- Using regular expressions instead of a custom parser means that parsing mistakes are not uncommon. The online version of the applet has a voting option enabled for particularly error-prone regular expressions, where more than one page must agree on the analysis for it to form part of the final graph. This eliminates some mistakes but at the cost of many valid intersections. In the future, the applet should make this behavior optional.
- Due to the regular expressions used, the application works only with English text and English street-naming conventions. When examining the output, it is also obvious that regional variations of English make a difference. American English, British English, and Australian English often have slightly different ways of referring to street relationships. I've tried to allow for some of the more common variations.
- **Google's 10-word limit** [["The 10-Word Limit" in Chapter 1](#)] means an inability to effectively filter out previously traversed street names by packing them into the query prepended by a negative (-) sign. Because the street traversal algorithm deliberately tries streets closest to the initial street, the search results returned on those streets often include pages already processed. These duplicate pages are filtered from further processing but getting search results back for already-processed pages wastes precious Google API key usage juice.
- Some streets are hard to disambiguate. Highly generic street names such as Main Street or High Street can pollute the connectivity graph. So, for example, if Interesting Road is found to intersect with High Street, there may be several High Streets found in the results other than the one coming off Interesting Road—thus producing connectivity via High Street, which is not even indirectly connected to Interesting Road and is therefore highly uninteresting.

Having more specific constraints in the search can help, but that will reduce the overall quantity of results.

## Google Cartography: Street Art in Your Neighborhood

Another approach would be to prune connections that have low connectivity with other parts of the graph. This will be the case with streets that come off the “wrong” High Street. Again, this would increase overall quality but at the cost of quantity, with inevitable false positives.

- The pattern matching (using regular expressions) for street name and type is fairly limited. Examples of streets the current pattern matching will not catch are Route N or Highway N and single name streets such as Broadway.

### Running the Hack

Point your web browser at <http://richard.jones.name/google-hacks/google-cartography/applet/mapping.html>.



You will need a recent version of the Java plug-in (1.3.x is not new enough). You can download the JRE which includes the java plug-in from <http://java.sun.com/j2se/1.4.2/download.html>. You'll also need your own Google API key [“Using Your Google API Key” in Chapter 9].

When the “Enter parameters” applet window appears (shown in Figure 3-7), enter your Google API key and adjust the maximums per instructions on the Google Cartography page. Now, type in your starting street and any additional search criteria by which to narrow the search.



Figure 3-7. Enter a starting point of interest and anything that might narrow down the possibilities

After a little churning, the applet will display a map for your street and region of interest similar to those in [Figures 3-5](#) and [3-6](#).



Be warned that the applet may use large amounts of bandwidth, depending on the parameters you enter.

—*Richard Jones*