Get Me to the Airport on Time!

Steven Foote

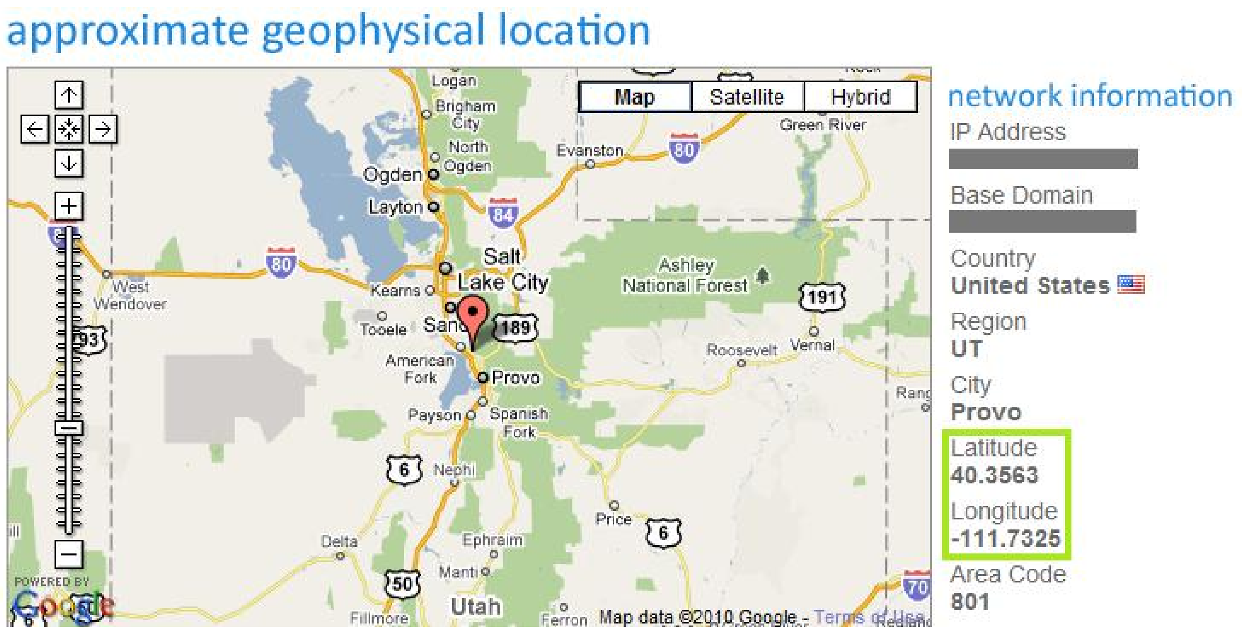
Executive Summary

When time is of the essence, it is sometimes very useful to have a computer aid us in our decisions. In approximately 1.5 months, my wife and I will become proud parents for the first time. The problem is that in less than one month, I will be starting a job in another state, and my wife will have to stay here in Utah - even if she were able to travel this late in pregnancy, her insurance will only cover her in Utah. With these precarious circumstances, in approximately 1.5 months I will need to take my flight (literally) as quickly as possible to return to Utah. However, as economical people, we would like to pay as little as possible for the flights to and from Utah. To this end, I have written a program to search for flights at five airports of my choosing, and finding the driving time to those airports from my current location. The program searches both the kayak.com and southwest.com websites for flight information.

Implementation Documentation

There were several problems that I identified that I would need to solve in order to make this program work. Most of these problems require VBA to access information on the Internet. My original hope was to be able to utilize APIs for Kayak and Southwest, but to my dismay, no such APIs are currently in existence. Therefore, I had to resort to scraping the HTML from these and other pages in order to effectively execute my program.

The first problem was in determining the users current location. While this would be quite simple if the application were running on a mobile phone, that is not the lot I have drawn. However, with the help of www.yougetsignal.com and Google Maps, this problem became relatively simple. First, VBA opens an (invisible) instance of Internet Explorer, and directs it to http://www.yougetsignal.com/tools/network-location/, which displays the following page:



network information

IP Address
[redacted]

Base Domain
[redacted]

Country
United States

Region
UT

City
Provo

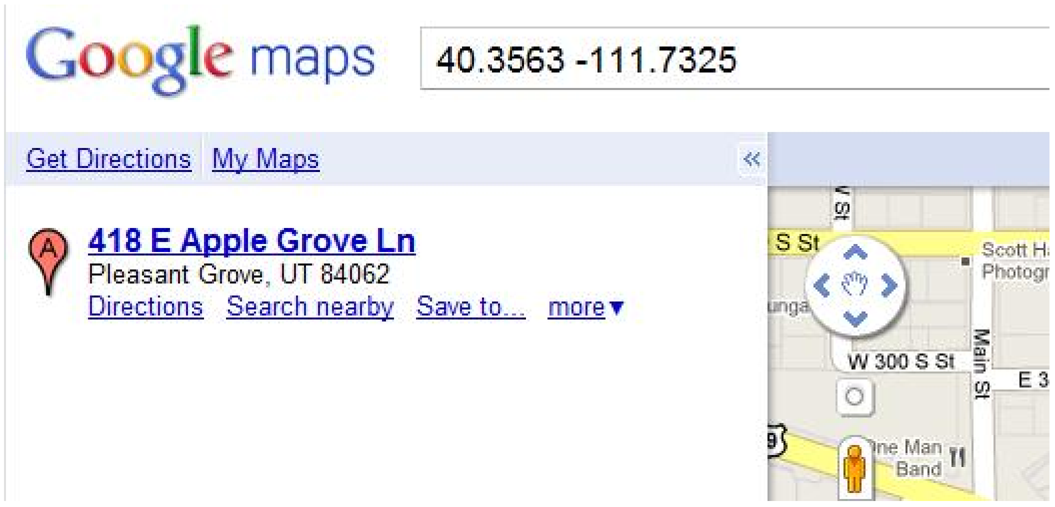
Latitude
40.3563

Longitude
-111.7325

Area Code
801

The default search performed at this URL is on the local machines IP address. The information found in the green box (exact latitude and longitude) is scraped from the page, and stored in VBA. www.yougetsignal.com approximates the geophysical location of a computer based on its IP address. As we will see below, approximation is the appropriate word.

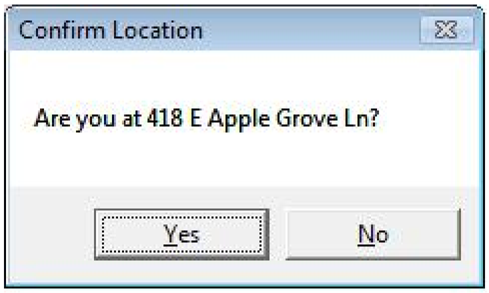
At this point, a new instance of Internet Explorer is opened by VBA, is directed to http://maps.google.com, and searches for the latitude and longitude coordinates found above. This search will reveal a more human readable location for the coordinates. For example, a search for the coordinates will give:



40.3563 -111.7325

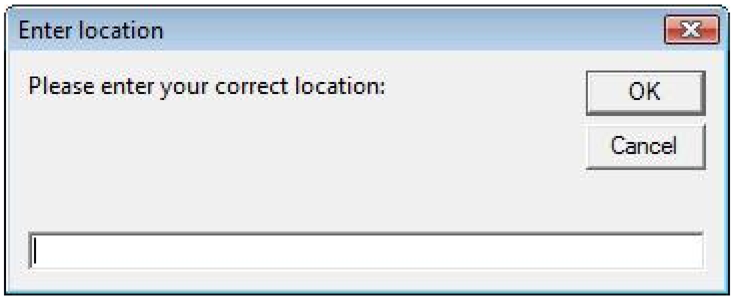
418 E Apple Grove Ln
Pleasant Grove, UT 84062

We can see that we have a problem, as this search was performed from BYU campus (which is 9.3 miles driving from the address above). The error is too large to ignore for our purposes, so VBA asks you to confirm your address before continuing:


Confirm Location

Are you at 418 E Apple Grove Ln?

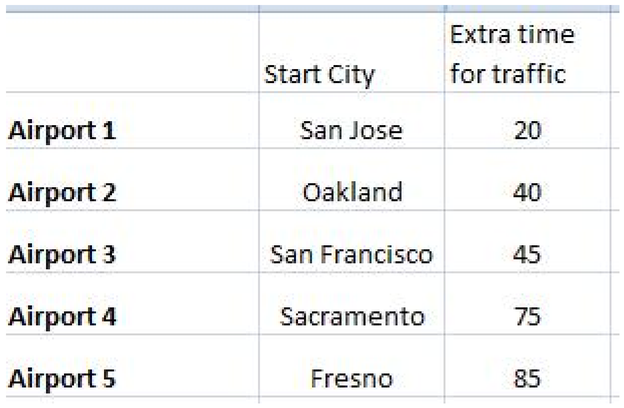
If the answer is no, then you are asked to enter your current address:


Enter location

Please enter your correct location:

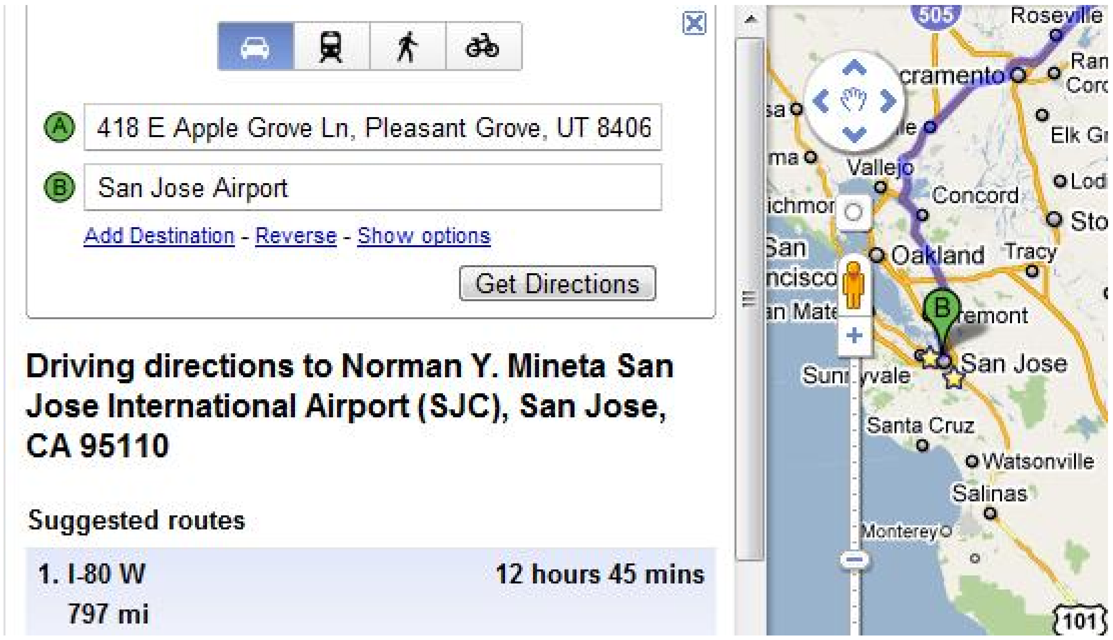
In this way, I do my best to programmatically determine the user's location, but when the IP address location fails, I allow the user to manually enter the exact location.

The second problem I encountered was determining when the earliest time I could catch a flight. Since the flight will be domestic, and I am an optimistic person, I approximated that I would need to be at the airport one hour before takeoff. Then I added a given number of minutes for traffic to each airport. This number can be changed in excel, but must be changed before the program runs (i.e. the program is not dynamic enough to adjust based on changes to these numbers after the program has run).



	Start City	Extra time for traffic
Airport 1	San Jose	20
Airport 2	Oakland	40
Airport 3	San Francisco	45
Airport 4	Sacramento	75
Airport 5	Fresno	85

The hardest part of this problem, determining the driving time to the airport, required more than just manually entered numbers in Excel. however, due to the magic of Google's natural language recognition in search, this part of the problem became very easy. Using VBA to navigate to and fill in the directions form on Google Maps, I can obtain the driving time to the airport.



A 418 E Apple Grove Ln, Pleasant Grove, UT 8406

B San Jose Airport

Add Destination - Reverse - Show options

Get Directions

Driving directions to Norman Y. Mineta San Jose International Airport (SJC), San Jose, CA 95110

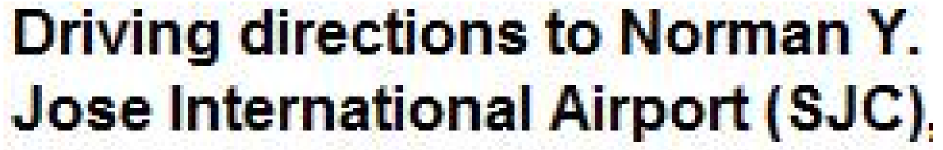
Suggested routes

1. I-80 W **12 hours 45 mins**
797 mi

Using the name of the departure city listed in Excel (e.g. San Jose, Oakland, etc.), I append the word "Airport", and Google's magic knows what I'm talking about. After waiting for the results to load, VBA finds the driving time (in the example above, 12 hours 45 mins), and converts that to excels time format (e.g. 12:45). Then, a formula in Excel adds the driving time to the current date and time (NOW()), plus traffic time, plus one hour (to arrive an hour early at the airport. Now, I have the earliest time I can catch a flight.

The last problem is in getting flight information. This turned out to be the most difficult, and certainly the most time consuming problem to solve. Initially I only wanted to search Kayak, which retrieves flight information for several airlines. However, I quickly became aware that Southwest beat the prices of all other airlines almost every time when it comes to same-day flight bookings. So, I decided to search both sites.

Conceptually, this is a very easy task to complete: open the site, execute the search based on origin and destination airports and time, search for flight times and prices, and put that data in the appropriate place in excel. However, HTML scraping is never an easy task, and few sites have messier HTML than travel/airline sites. While Google elegantly handles natural language in its searches, kayak.com and southwest.com scoff at such an idea. If you don't search by airport code, you are out of luck (usually - searching for "Salt Lake City" actually works on kayak.com). Again, Google to the rescue - note the text in the previous screenshot. Here's a zoomed version of the important part:



**Driving directions to Norman Y.
Jose International Airport (SJC).**

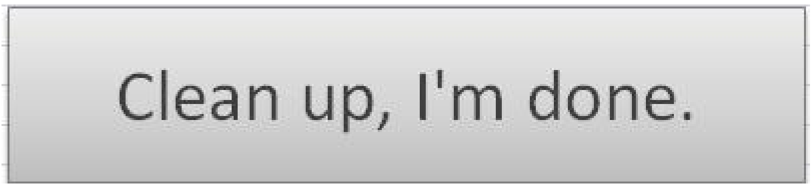
Note that placed conveniently between the parentheses is the airport code. Wow. So, VBA collects that information from the Google Maps directions for each airport, and stores it in an array for use in searching kayak.com and southwest.com.

Using the Cells().Text property, VBA retrieves the date and time to use in the search. I quickly realized, though, that if I try to search for a specific hour on kayak.com, I rarely get any results (especially if that specific hour is some time today). So, instead I search for a time range on both sites.

Once the search has been executed and the results page has loaded, VBA finds the takeoff time, arrival time, and price of the first flight in the list. On southwest.com, if a flight is found but it is not available (sold out or otherwise unavailable), the corresponding cells in Excel are filled in with the words "no" "such" "luck". A lack of results or other errors on kayak.com are harder to spot, and as such are handled much less gracefully.

Due to the potential errors in the less than perfect scraping of the results pages, there needs to be a quick way for the user to quickly view the actual search pages. Because the program creates 12 distinct instances of Internet Explorer as it runs, it would be unwise to have all of these windows be visible as the program runs. Clutter is bad. So, instead I set an event on each of the cells where price is held similar to the onFocus event in HTML/JavaScript. When the selected cell on the sheet changes, the procedure checks if it is one of the "Price" cells. If it is, it checks the row and makes visible the Internet Explorer that corresponds to that row. This Explorer will contain the search results for that particular row, so the user can see the full set of results, instead of just the top result. This allows the user to quickly select and book the desired flight, as well.

Once the user is done with the running the program and selecting a flight, s/he can click



to close all of the instances of Internet Explorer (though they may be invisible), and clear the cells on the worksheet that contain results.

Instead of taking valuable time (in my case, extremely valuable time) opening several tabs in a browser, and manually searching all of the possible flights, this program executes all of the searches for me, and does so quite quickly, summarizing what are likely the most promising results in a way that is easy to compare. This should save me a lot of time, when time is of the essence.

What I Learned

Before beginning this project, I already had some experience working with webpages, using the DOM and JavaScript. I thought I would have an easy time in this project. I quickly learned that doing similar work in different programming languages is not to be taken lightly. Accessing and manipulating HTML in VBA is similar in many ways, but different enough from JavaScript that the project quickly became quite difficult.

Additionally, I have never had to scrape pages before. I have inspected the HTML of sites from Gmail to Facebook to Twitter, to try to understand how they accomplish certain bits of styling, but I haven't ever tried to pull data directly from their HTML. That is why I was hoping I would be able to pull data from Kayak and Southwest using an API - parsing data that is sent to be parsed (e.g. XML or JSON) rather than data that is meant to be displayed (HTML) is much easier. Scraping was difficult, time-consuming, and humbling. And the worst part is that, if any of the sites I have scraped change their HTML, even just a little bit, it could completely break my code. Even as it is, the results retrieved are not always perfect.

In my programming experience, I have occasionally used the work of others, in the way of libraries and pre-written classes, etc. However, I usually prefer to write my own code because I know, if it breaks, it's my fault. I know what it does and how it works, because I wrote it. However, I recently began (cautiously) using the jQuery library in my JavaScript code. In this project, I learned more fully the value of "standing on the shoulders of giants." As I was trying to access the HTML from the Kayak results page, the agent class (used for accessing websites, written by Professor Gove Allen) broke. My first instinct was to try to figure out how to access Internet Explorer on my own - to bypass entirely the work done by my good professor. Instead, however, I sought out his help, the fix was very easy, and my code became much easier to write. Had I attempted to "recreate" the agent class, I would have probably doubled the time I took on my project.

Finally, I realized that sometimes you have to limit what your program will do. With enough time, I could have probably booked and paid for my flight all from excel. I could make everything perfect, and spend the rest of my life doing it. But, the purpose of this project was to help me make a quick decision about which flight to take. So practically, this program will only ever be run once for its originally intended purpose. I learned that limiting oneself to one's scope is a very important aspect of effective programming. I would not want my life's work to be an Excel Spreadsheet that gets flight information, even if it does so really really well.