ACERCAMIENTO AL USO DE LA PRIVACIDAD EN INTERNET PARA LA DISCRIMINACIÓN DE PRECIOS

Manuel Soto Jiménez

Universidad Autónoma de Madrid

Fecha: 29 de Abril de 2019

Indice

1

Introducción

A continuación se expondrán los objetivos principales de este trabajo, así como el planteamiento del abordaje hacia dichos objetivos.

La discriminación de precios

En Economía, se denomina **discriminación de precios** al acto de un vendedor de cobrar distintos precios a consumidores por un mismo producto. La palabra discriminación quizás pueda inducir a pensar en "ilegal", pero lo cierto es que en absoluto (salvo excesivos casos) hay ilegalidad en este acto. Distinguimos tres grados de discriminación de precio:

- **Tercer grado**: Es la más usual. Reside en cobrar distinto precio en función del grupo de consumidor. Es decir, tenemos discriminación de tercer grado cuando, por ejemplo, cobramos la entrada a un museo según seas estudiante, niño/a, jubilado/a, etc.
- **Segundo grado**: Tiene lugar cuando es el propio consumidor el que se selecciona. ¿Cómo es posible que un consumidor, pudiendo elegir, no elija el grupo

más barato? Un ejemplo que contesta a esto ocurre con los descuentos por cantidad, ya que no siempre queremos la mayor de las cantidades a comprar.

- **Primer grado**: Es el principal en el que nos vamos a centrar. Se le llama también **discriminación perfecta**, y está basada en cobrar un precio distinto a un consumidor específico, con el fin de maximizar la ganancia en caso de venderle a dicha persona nuestro producto. Un ejemplo de esto podría ser un casero que nos suba el alquiler a sabiendas de que queremos estar en este piso; o bien un único médico de un pueblo, que sabe que es el único recurso para la salud de estos habitantes.

Acabamos de profundizar en el concepto de discriminación de precios con el objetivo de responder a la siguiente pregunta, que es la temática de este trabajo:

La pregunta

> **¿Las aerolíneas principales del mercado aplican discriminación de precios de primer grado a sus usuarios de Internet?**

Que nos aplican los otros dos grados ya sabemos que es cierto: Elegir viajar en clase business, tener descuentos por número de billetes, etc. son ejemplos de dichos grados. Sin embargo, como bien indica la pregunta principal, vamos a centrarnos en el primer grado.

En otras palabras, si bien está claro que una aerolínea va a subir/bajar el billete de un vuelo en función de distintos análisis de mercado (de entre los cuales iremos viendo que Internet influye bastante y en su mayoría), la duda queda en el ámbito personal. Si al lector no le ha ocurrido lo siguiente, es probable que conozca a alguien que sí: Llevas x días vigilando un vuelo para tus vacaciones, intentando coordinar las horas de salida y llegada con el resto de compañeros de viaje, comparando precios, buscando nuevas ofertas… y finalmente te decides por comprar ya. Vas a buscar por enésima y última vez tu vuelo y… ¡Sorpresa! De repente acaban de subirte el precio.

A veces a la aerolínea le sale mal este movimiento: eres bastante testarudo y al final consigues otro vuelo en otra compañía. Sin embargo, la mayoría de veces acabamos cediendo, porque probablemente siga siendo la mejor opción. Es ahí cuando triunfa

este primer grado. Acaban de apropiarse de casi todo nuestro excedente por este vuelo. ¿Es esto casualidad? ¿Tuvimos la mala suerte de comprar justo cuando fluctuó el precio o, efectivamente, tuvimos que haber navegado las (n-1) veces anteriores en modo de incógnito?

Investigaciones anteriores

El primer paso de una investigación consiste en ver si la respuesta ya está argumentada. Por supuesto, dado que esto no es un trabajo bibliográfico, se muestra a continuación que hay duda en este tema. De todas las noticias que intentan dar luz a nuestra pregunta, vamos a ver los dos polos opuestos.

En un primer lugar, TheSun (periódico de UK) nos dice que **no**: Nos introduce una explicación a lo que son las **cookies** (de las que hablaremos más tarde) y entrevistan a otros reporteros que investigaron en el asunto. Hicieron la prueba por su cuenta y concluyeron:

> "We put it to the test and found… that it doesn't make a difference"

Por último nos traen una serie de consejos para hacer nuestro vuelos más baratos como premio de consolación.

Ahora vamos con Time, que contesta a la pregunta con un **sí**: William McGee, un asesor de aeronáutica, realizó un estudio junto a un equipo. Buscaron mismos vuelos en nueve aerolíneas diferentes con dos navegadores: uno con sus cookies intactas, el otro permitiendo el uso en todas.

> "McGee found that 59% of the times when the searches differed, the fares were higher on the scrubbed browser — the browser with no search history"

¿Cuál es el problema entonces, si ya tenemos respaldo científico sobre esta pregunta? Ocurre que el total de búsquedas fue de 379. McGee se juntó con un equipo de investigación para realizar "a mano" todas esas búsquedas. Son bastantes cuando hablamos de "a mano", pero insuficientes para respaldar una respuesta.

Objetivo del trabajo

Aquí es donde entra este trabajo. Ya sabemos qué queremos demostrar, y este es el esquema a seguir para acercarnos a la respuesta:

- **Leer sobre qué afirman las aerolíneas al respecto**. Fijaremos el marco de investigación en algunas de ellas.
- Con nuestras aerolíneas objetivo, **analizar el tráfico que intercambiamos en un proceso normal y habitual de búsqueda de vuelos**. Esto nos dará una idea sobre qué mensajes (y qué campos del mismo) en dicho proceso son clave para quizás inducir a discriminación de primer grado.
- **Implementar una librería** que realice de forma automatizada las peticiones que hemos destacado del punto anterior. Diferenciaremos las distintas formas de poder realizar la búsqueda en lo que respecta a privacidad.
- Con las respuestas recibidas, mejorar la librería para que pueda parsear los datos objetivo de las peticiones (es decir, guardar el precio y detalles del vuelo). Básicamente **vamos a implementar una Araña Web (o *spider*) de búsquedas de vuelos**.
- **Recopilar todos los datos obtenidos y sacar conclusiones**.

2

Legalidades: ¿Qué puedo y qué pueden hacer?

Pese a que este apartado no sea puramente Ingeniería Informática, lo considero de vital importancia no solo para este trabajo. Vamos a dar respuesta a las siguientes dos preguntas:

- ¿Tengo derecho a investigar el funcionamiento del ajuste de precio de vuelos de una aerolínea?
- ¿Hasta qué punto puede una aerolínea saber de mí con el objetivo de ajustar el precio de sus vuelos?

Propiedad Intelectual

La respuesta a la primera pregunta está en la **Ley de Propiedad Intelectual** (de ahora en adelante LPD). La Ley de Propiedad Intelectual es un Real Decreto que data de 1996. De entre sus 203 artículos, destacamos el Título VII: "Programas de Ordenador".

Descubrir cómo cambia el criterio de una aerolínea para la discriminación de precios supone un proceso de ingeniería inversa. Si bien no se podrá obtener el código fuente de ningún programa, al fin y al cabo vamos a realizar pruebas con tal de saber cómo funcionan los mismos. Esto puede (pero veremos que no para nuestra finalidad) vulnerar el derecho de quienes hayan programado toda esta estructura.

El artículo 100.3 de la LPI es el que recoge nuestra respuesta:

> "3. El usuario legítimo de la copia de un programa estará facultado para observar, estudiar o verificar su funcionamiento, sin autorización previa del titular, con el fin de determinar las ideas y principios implícitos en cualquier elemento del programa, siempre que lo haga durante cualquiera de las operaciones de carga, visualización, ejecución, transmisión o almacenamiento del programa que tiene derecho a hacer."

Dado que "tenemos una copia" de los programas que vamos a observar y vamos a usarlo de forma lícita (no valdría aprovechar fallos o modificarlo para realizar investigación), sin que nos tenga que autorizar nadie **podremos ponernos manos a la obra**.

Este era el artículo más importante a destacar, pero mencionamos ahora algunas limitaciones y extensiones:

- El 100.3 es válido salvo en excepciones recogidas en el 100.5 y 100.6. Un ejemplo de una de estas limitaciones es el de investigar el programa con tal de comercializar una copia de este (100.6c).
- Si además el proceso de ingeniería inversa es de fin educativo está a su vez respaldado por el artículo 32 de la LPD.

Protección de Datos

Es claro que las aerolíneas no inflingen actualmente contra leyes que amparan nuestra privacidad. No se entrará en detalle sobre legislaciones como la Ley de Protección de Datos española, o "The Cookie Law" (de la Unión Europea). Cierto es que a continuación mencionaremos sobre qué sostienen algunas aerolíneas sobre cómo utilizan nuestra privacidad.

Como ejemplo hablaremos sobre la aerolínea española **Iberia**, que contiene la

respuesta a nuestras dudas en su sección de "Información Legal", en "Política de protección de datos personales". Se nos expone la información a modo de pregunta y respuesta, de las que destacamos:

- **¿Cuándo recabamos tus datos personales?:** Con cada vez que utilizamos los servicios de Iberia, nuestros datos personales son recogidos de distintas formas y para diferentes usos. En relación con la temática del trabajo:
  - *"Cuando reservas o buscas un vuelo u otros productos o Servicios en nuestra web o en nuestra aplicación móvil"*. Es decir, nos da un indicio bastante probable de que (como era de esperar) información sobre nosotros se nos recaba en el momento de utilizar el buscador de vuelos. Cierto es que no se especifica si es con un objetivo de ajustar el precio a nivel general o individual (nuestro objetivo).
  - *"Empresas que nos proporcionan datos bajo políticas de privacidad que prevén información a compartir con Iberia."*. Este dato es bastante importante. Por si el lector lo desconocía, **no buscamos en el anonimato cuando utilizamos buscadores de vuelos**, o al menos en un carácter general. Por ejemplo, si buscamos en una página como Kayak, esta tiene derecho (también figurará en su política de de protección de datos personales) a comerciar con Iberia el hecho de que X persona está interesada en X vuelo. Lo que desconocemos, otra vez, es si esa información se usa a nivel individual o a nivel general.
- **¿Qué tipos de datos personales recabamos y conservamos?:** A destacar la sección *"Datos sobre el uso que hagas de nuestros sitio web, aplicaciones para móvil y centros de atención al cliente."*. Iberia menciona:

  A partir de tus datos de uso podremos saber que has visitado iberia.com y buscado un vuelo.

En resumen, en el caso de Iberia (y de la mayoría de aerolíneas) tenemos constancia de que **el acto de buscar un vuelo es coleccionado por la aerolínea**, pero nos falta el punto clave de si dicho acto se usará en un carácter general o, por el contrario, se nos "personalizarán" nuestras ofertas de acuerdo con lo buscado anteriormente.

Destacamos unos últimos aspectos sobre nuestra privacidad y qué pueden hacer las aerolíneas con ella:

- La Unión Europea ampara nuestra privacidad con lo que se conoce como **"derecho al olvido"**. Es decir, una empresa que hace uso de nuestros datos debe borrarlos en caso de que nosotros se lo pidamos, salvo que dichos datos sean necesarios en ese momento para otros fines jurídicos. Si se investiga bien en las secciones de política de privacidad, por ley debe haber un formulario u opción mediante la cual se retira nuestro perfil de las bases de datos de publicidad (de las aerolíneas por ejemplo).
- Comprobando desde el otro lado de la ecuación, los buscadores de vuelo afirman en sus políticas de privacidad que envían agregados de datos a otra aerolíneas. Un detalle importante de esto es que estos agregados de datos **son anónimos**. Citando, por ejemplo, a **Skyscanner**:

  "[…] tomar decisiones fundamentadas y estratégicas relacionadas con la empresa mediante la creación de conjuntos de datos agregados y anónimos que nos permitan generar estadísticas útiles y convertirlas en patrones o tendencias dentro del sector de los viajes. […] Esta información también es útil para los Proveedores de viajes, ya que les permite identificar la demanda de nuevas rutas o destinos y ofrecer tarifas competitivas."

Síntesis

Ahora tenemos unas "reglas" de cara a investigar:

- Buscar tanto en aerolínea como en buscador de vuelos no nos salva de formar parte de un estudio de mercado (esto es obvio y se sale de la discriminación de primer grado).
- Buscar en aerolínea no precisa el hecho de ajustarnos precios de forma individual.
- Buscar en un buscador de vuelos nos garantiza el anonimato en lo que respecta a un "se dice el pecado, pero no el pecador". (Ojo: este anonimato puede perderse al momento de redireccionarnos el buscador a la web de la aerolínea)

Antes de pasar a la siguiente parte, fijamos ahora el marco de investigación:

- **Aerolínea 1:** Iberia
- **Aerolínea 2:** Ryanair

3

Primer Contacto: Tráfico Usual

En esta sección, analizaremos los paquetes HTTP que enviaremos a los servidores de las aerolíneas y buscadores fijados. Para ello, hacemos uso de **Burp Suite**, una herramienta muy potente de análisis de tráfico de red.

Burp Suite

Se ha elegido Burp Suite en lugar de Wireshark (de momento) por el hecho de que Burp Suite puede "paralizar" el proceso de comunicación. En resumen, su funcionamiento es el siguiente:

- Vinculamos un proxy desde Mozilla Firefox hacia nuestro propio ordenador (loopback), es decir, todo paquete que quiere salir a Internet antes pasa por mí mismo.
- Burp Suite se vincula a este proxy para leer cada paquete que salga. Se almacena la petición y la respuesta e incluso se da opción de guardar un archivo con los paquetes intercambiados (hasta el momento actúa igual que Wireshark).

- La diferencia principal, **Burp Suite puede parar el mensaje HTTP y dejarnos elegir su envío o no.** Podemos incluso modificar el paquete que estamos a punto de enviar. De la misma forma, cada uno de estos paquetes se almacena y se puede guardar como un archivo.

De esta forma vamos a situarnos justo antes de realizar una búsqueda en nuestras aerolíneas y sacar conclusiones de lo enviado.

Petición POST: Iberia

El archivo `post_iberia` consiste en una serie de objetos XML que mandamos guardar a BurpSuite para poder analizarlos. Este archivo (junto a los demás que estudiemos) consiste en una serie de datos sobre el archivo de guardado, y posteriormente items XML donde cada uno es una petición respuesta HTTP. Con ayuda de `grep` y otros comandos podemos pasar a estudiar datos sobre este proceso de búsqueda de vuelo.

Hosts con los que comunicamos

Una búsqueda en Iberia está formada por 84 paquetes HTTP:

Host	Paquetes
www.iberia.com	24
decollector.tealeaf.ibmcloud.com	17
krxd	7
attribution.iberia.com	5
criteo.com	5
api.flocktory.com	5
service.maxymiser.net	3
bat.bing.com	3
www.google-analytics.com	2
www.googleadservices.com	2
www.facebook.com	1
stats.g.doubleclick.net	1

Host	Paquetes
www.google.com	1
bucket.cdnwebcloud.com	1
ayuda.iberia.com	1
s3.eu-central-1.amazonaws.com	1
platform.twitter.com	1
static.ads-twitter.com	1
analytics.twitter.com	1
push.services.mozilla.com	1

Tabla 1: Hosts destacados de una petición a Iberia

Salta a la vista la cantidad de hosts "ajenos" a Iberia a los que acabamos de enviar información por habernos interesado en la búsqueda de un vuelo. **Solo un 36,7% de los mensajes (30/84) que enviamos en una búsqueda de vuelo van destinados a Iberia**. Algunos de los hosts "intrusos" son conocidos a nivel casual: Facebook, Google, Bing, Twitter… Otros tantos son hosts de nombre autoexplicativo: el Ad Services de Google, Google Analytics, Twitter Analytics. Todos estos mensajes que enviemos, muy probablemente, afectarán al hecho de la publicidad que nos aparezca la próxima vez que naveguemos por la web (ahora no paran de anunciarme vuelos en cualquier página web).

Sin embargo, vamos a investigar quiénes son algunos de los hosts desconocidos:

- `decollector.tealeaf.ibmcloud.com`: Introduciendo este enlace directamente en un navegador web, nos aparece una interfaz de Mozilla que analiza un código de respuesta de error que nos ha devuelto el host. Este host proviene de Tealeaf, una herramienta de IBM de análisis de datos movido por inteligencia artificial con propósito de marketing.
- `krxd`: Es un host que pertenece a Krux Digital, una compañía estadounidense de (otra vez) análisis de datos para generar un `profile` de intereses de los visitantes. Este host aparece en una cantidad inmensa de páginas web al ser una de las herramientas de análisis más utilizadas. El lector puede probar con cualquier página web comercial y analizar los paquetes para encontrarse con este dominio (muy probablemente).

- `criteo.com`: Otra corporación de análisis de datos destinada a fines publicitarios y de comercio perfilado. Compañías como XBox, P&G, Philips, Reebok… son clientes de esta empresa.
- `api.flocktory.com`: Otra compañía de marketing. Con afiliados como Media Markt, Worten, la ONCE, DIA, Burguer King…
- `service.maxymiser.net`: Este host como tal forma parte de la API de Maxymiser, el cual es un servicio de Oracle de publicidad, análisis de datos, etc.

Resulta sorprendente la gran cantidad de hosts (y múltiples mensajes) con los que conectamos en el proceso de vuelo cuya finalidad es únicamente la de analizar nuestros datos para publicidad y economía. Queda bastante claro que buscando de forma normal pasamos a intervenir en estudios de un mismo mercado realizados por casi una decena de empresas distintas: Google Analytics, Facebook, Twitter Analytics, Tealeaf, Krux Digital, Criteo, Flocktory y Maxymiser.

Como experimento, probamos a realizar la misma búsqueda anterior pero bloqueando ciertos hosts (desde BurpSuite, deniego los paquetes que vayan a ser enviados) con tal de ver cuáles de ellos son realmente necesarios para saber sobre lo buscado. **La búsqueda funciona pese a bloquear todo aquel host que no sea www.iberia.com y attribution.iberia.com**.

Uso de Cookies

Es sabido lo que es una Cookie: pequeña cantidad de información que almacenamos (cuando un host lo pide con `Set-Cookie`) para posteriormente ser usada en otras ocasiones al comunicarnos con un host (que mostramos mediante el header `Cookie`). Ahora veamos qué puede llegar a hacer esta pequeña información en la vida real.

De todos los paquetes, se destaca el que contiene un POST a `SearchFormHome.do`. El funcionamiento de la búsqueda en Iberia en lo que respecta a estructura de su página web se basa en dicho recurso. De este mensaje, que es el más importante, analizamos las cookies que contiene el mismo.

```
Cookie: popunderConf=1;
```

_fbp=fb.1.1553023523802.833910812;
_ga=GA1.2.2143141733.1553023524;
IBERIACOM_LAST_REQUEST=MAD|VGO| | | |;
IBERIACOM_LAST_SEARCHS=Madrid (MAD)#MAD#Vigo
(VGO)#VGO#31/05/2019#07/06/2019#R###INF-0|ADT-1|YTH-0|CHD-0|YCD-0;
__utmx=206024267._I8wQFOkTreS-jZYz2-Gow$0:;
__utmxx=206024267._I8wQFOkTreS-jZYz2-Gow$0:1553285810:8035200;
_gcl_au=1.1.922693830.1553023681;
flocktory-uuid=1104d3bc-9e0b-4191-9ff3-b7a96af22acb-1;
cto_lwid=eec87909-0f16-41b6-8719-a441fb4d508f;
cto_idcpy=b1921301-cee7-4c0c-a479-bb2987464133;
IBERIACOM_IDENTIFICATION=155303599887923474369578103697 91;
IBERIACOM_PERSONALIZATION=ES|ES|es|MAD|VGO|ES|;
mt.v=2.420654002.1553158572927;
JSESSIONID=l7WnChFybLjGjzjEtrTt0Op9CukZgcf3j3MVlMiNuCiyy1kD3Aqv!-1889617234;
BIGipServerwww.iberia.com_SSL=1897638080.47873.0000;
rxVisitor=1553285650730KHHFITTJSESOG0M00JPCET00N43BAG0B;
dtPC=4$485805725_708h-vJAFNENAPDHPKKPLDIFPBPGAMHCADPEKO;
rxvt=1553287625748|1553285650732;
dtSa=-;
dtLatC=12;
IBERIACOM_COOKIES_ALERT=shown;
WCXSID=55536052583953490829052791 66;
TLTSID=00005553605258395349082905279166;
_gid=GA1.2.849639312.1553285654;
PC_Passengers=;{"adult":1,"children":0,"infant":0};
mmapi.store.p.0={"mmparams.p":{"pd":"1584821719106|\"509263152|GA[...]AAAUU=\"",
"srv":"1584821719110|\"fravwcgeu09\"",
"uat":"1584821744482|{\"Roundtrip2plus\":\"No\",
\"Market\":\"ES\",\"Language\":\"es\",\"Log_Status\":\"Not_loggedIn\",
\"Quadrigan\":\"IBHMPA\",\"Analytics_Code\":\"es/ES/IBHMPA\",\"PAXNum_S\":\"Single\",
\"Origin\":\"MAD\",\"Destination\":\"VGO\",\"LeadTime\":\"90days\",
\"FlightType\":\"RT\",\"LOT\":\"6to8\",\"Flight_Search\":\"MAD,VGO\"}"},
"mmparams.d":{},"price_alerts_CD":{}}

```
mmapi.store.s.0={"mmparams.d":{},"mmparams.p":{}};
ga_flight_totalvalue=;
    ga_flight_originid=MAD;
    ga_flight_destid=VGO;
    ga_flight_startdate=2019-05-31;
    ga_flight_enddate=2019-06-07;
dtCookie=4$850F655A678ACFF6C774930C4316B786
```

Texto 1: Cookies traducidas en la petición principal de Iberia. Se han omitido códigos excesivamente largos y se ha traducido del URL Encode.

Destacamos lo siguiente de las cookies que fueron enviadas:

- Como era de esperar, se envía con nuestras cookies (no únicamente a Iberia) el vuelo que acabamos de consultar con todo detalle e incluso las búsquedas anteriores. Esto puede verse en `IBERIACOM_LAST_REQUEST`, `IBERIACOM_LAST_SEARCHS` y en las de `ga_flight`, que pertenecen a Google Analytics.
- Más destacable es el hecho de que **guardamos ciertas cookies que contienen un ID de usuario e incluso de la sesión**. Se puede comprobar con, por ejemplo, `flocktory_uuid`: *"Identifier of the user"* según describen en su política de uso de cookies. Realizando el mismo experimento desde otros ordenadores, vemos que dicho identificador varía pero es fijado por cada "máquina" salvo que borremos las cookies.

Petición GET: Ryanair

Los objetivos de analizar la petición GET de búsqueda de un vuelo con Ryanair son los siguientes:

- Comprobar que Iberia (como es de esperar) no es la única aerolínea que envía información nuestra y distinguible a otras empresas de publicidad y análisis.
- Comprobar que buscar un vuelo con el verbo GET es más simple (de cara a ahorrar dificultad en la siguiente fase de investigación).

Hosts con los que comunicamos

Host	Paquetes
www.ryanair.com	24
app.launchdarkly.com	4
d.adroll.com	3
events.launchdarkly.com	2
clientstream.launchdarkly.com	2
ryanair.tt.omtrdc.net	2
desktopapps.ryanair.com	2
services-api.ryanair.com	2
assets.adobedtm.com	1
widget.criteo.com	1
sync.outbrain.com	1

Tabla 2: Hosts destacados de una petición a Ryanair

Vemos que en cierto modo, Ryanair es "más limpio" en lo que respecta a con quién nos comunicamos. **De 44 paquetes, 30 son de Ryanair (68,2%).** Analicemos los hosts restantes:

- `launchdarkly`: Es una empresa de *feature flags*. En resumen, una *feature flag* es una sección de código que puede estar o no dependiendo únicamente de una *flag*. Esto es de gran valor para el software de una empresa, dado que pueden realizar pruebas sin temor a provocar fallos en el resto de código (y con ello de pérdidas). Una gran práctica de este recurso es el conocido como **A/B testing**, que consiste en tener dos versiones de una misma página web o aplicación para poder estudiar cuál de las dos versiones genera más beneficios. Es de esperar que Ryanair hace uso de Launchdarkly para *A/B testing*.
- `d.adroll.com`: Volvemos a las empresas de marketing y análisis de mercado.
- `assets.adobedtm.com`: Es la herramienta de analísis y marketing de Adobe.
- `criteo` y `outbrain`: Otras dos empresas de análisis de mercado y marketing.

Uso de Cookies

Las cookies utilizadas por Ryanair como tal (es decir, conectando con el host `www.ryanair.com`) son pocas si bien indescriptibles:

```
s_nr2=1555499120838-Repeat;
agsd=0FZK3Jw_F3w6CFB61eydrLe6BJ8Xv2eeORgSxGRvAuRvfnN-;
AMCV_64456A9C54FA26ED0A4C98A5@AdobeOrg=-330454231|[...]|vVersion|3.1.2;
mkt=/gb/en/;
mbox=PC#b905573f3de84f128c828f8623[...]|session#[...]8d1e890#1555501103;
RYANSESSION=XLcF8golArkAAEwqxrAAAAAB;
check=true;
fr-correlation-id=fca328ac-8ef0-4c61-ad51-3f7c82b54511;
AMCVS_64456A9C54FA26ED0A4C98A5%40AdobeOrg=1;
s_cc=true; agso=AQ9ubMoBAPJIwikkw9ZItPmi_qiDgi8.;
agsn=iVuYdLXpKNnyt0HfFOAquzT-3uYeIzjjmsdvJdK-Nv0.;
agsy=ARQlAh4BAAxDFdkkw9ZIKNlONg..;
s_sq=%5B%5BB%5D%5D
```

Texto 2: Cookies traducidas en la petición principal de Ryanair. Se han omitido códigos excesivamente largos y se ha traducido del URL Encode.

No es trivial descifrar estas cookies en base 64, pero no supone problema alguno puesto que no son las únicas utilizadas. A diferencia de Iberia, hacemos uso de distintas cookies para distintos hosts en lugar de tener la mayoría de ellas en la petición principal de búsqueda. Vemos que, por ejemplo, al intercambiar tráfico con Criteo se comparte una única Cookie:

```
uid=b1921301-cee7-4c0c-a479-bb2987464133
```

Texto 3: Cookies en una petición de Criteo

Análogamente al caso de Iberia, vemos que disponemos de un identificador único para nosotros a la par que estamos enviando información del vuelo que estamos buscando.

Falta un último rasgo de las redes de comunicación entre nosotros y los hosts de aerolíneas que hemos de tener en cuenta de cara a responder a nuestra pregunta principal.

Una Última Herramienta de Marketing

No todo es la capa de aplicación HTTPS. A nivel de capa de red, tenemos un identificador único distinto a los `uid` que encontrábamos en ciertos paquetes: **Nuestra dirección IP, lo cual da lugar al IP tracking**.

El *IP tracking*, tal y como su nombre indica, es llevar cuenta de las direcciones IP que han contactado con nosotros. Esto aporta la gran ventaja de acabar con el anonimato de una conexión desde el nivel de capa de red. Que una aerolínea aplique *IP tracking* o no es cuestión de acceder a su código fuente, pero es de esperar que lo haga debido a las grandes ventajas que esto conlleva. Nos conducimos a destacar dos aspectos de esta técnica en el contexto de este trabajo:

- **NAT (Network Address Translation):** Estar detrás de un router supone, probablemente, que nuestra IP como tal no sea trackeada, puesto que esta es privada. Esto es ventaja y desventaja a la vez: Un host que haga uso de *IP tracking* no sabe que hemos sido exactamente nosotros quienes estamos interesados en X vuelo, pero probablemente en *IP tracking* "se corte por lo sano", de forma que entre todos los de una misma subred se están perjudicando (siempre y cuando la pregunta final sea cierta, claro).
- **VPN (Virtual Private Network):** Conectarnos a través de una VPN con un host que nos trackea hace que no nos rastree directamente. Sin embargo, repetir búsquedas desde dicha VPN tiene como consecuencia el mismo efecto, puesto que sería claro que estamos interesados.

Más adelante, veremos que es posible cambiar de IP constantemente y disipar los efectos del *IP tracking*.

Síntesis

Abandonamos la fase de análisis de tráfico usual con las siguiente premisas claras:

- Intercambiamos bastante información con hosts innecesarios de cara a obtener los resultados de búsqueda. Bloquear dichos dominios parece ser ideal para no formar parte de decenas de diversos estudios de mercado.
- Tenemos certeza de que existen Cookies en el tráfico usual con información del vuelo a buscar y de búsquedas anteriores; así como otras para empresas de marketing y análisis de datos que nos identifican únicamente. Hay incertidumbre en el hecho de si dicho identificador único sirve para una discriminación de primer grado.
- No sabemos si las aerolíneas hacen uso de *IP Tracking*, pero comentadas sus ventajas (desde el punto de vista de una empresa), es probable que sea así. Esta puede ser otra forma de identificarnos únicamente.

4

Implementación: Requests

Nota: Todo el código de este trabajo está hecho en Python3. Lo he elegido debido a su versatilidad, rendimiento y, más importante, simpleza de cara a las funcionalidades a implementar.

En `flight_search/requests` se ubican esbozos de código de cómo buscar vuelos utilizando `requests`. Hablo de esbozos puesto que más adelante veremos que surgieron varios problemas.

HTTP Requests

`requests` es el paquete de Python más recomendado para realizar peticiones HTTP. Su facilidad en los argumentos de entrada para los métodos `GET` y `POST`, así como en el retorno de dichas funciones (que es el recurso pedido en texto plano), son motivos para implementar una *spider* con `requests`.

Desde el punto de vista de nuestro trabajo, usar `requests` para buscar un vuelo es "canónico" por diversos motivos:

- Introduciríamos el mínimo de datos necesarios para obtener información de un vuelo.
- Podríamos elegir si enviar las cookies o no añadiéndolas como argumento al `GET` o al `POST`.
- Solo pasamos por los dominios justos y necesarios, sin terceros que se encarguen del marketing.

En el papel se ve bastante bien la planificación del trabajo, pero la realidad es que nos encontramos con un primer problema al intentar sintetizar nuestro tráfico con Iberia y Ryanair.

Primer Problema: Peticiones POST

Intentando empezar la fase de implementación con Iberia, vemos que la URL clave es `https://www.iberia.com/web/dispatchSearchFormHOME.do`. La diferencia con Ryanair es el hecho de que se hace *request* de dicha URL con un `POST`. Esto no supondría problema si no fuera por lo bastante "modelo de caja negra" que es dicha petición. Nos encontramos con una serie de argumentos indescifrables, parecidos a cuando analizábamos algunas cookies (consultar `flight_search/requests/iberia_params.txt`).

A continuación se muestran las funciones mediante las que realizábamos la petición `POST`, cuyos parámetros se leían de un `.txt` (incluido en el apéndice el ejemplo de Iberia):

```
def simple_post(url,params):
    """
    Attempts to send a post http request with the params given,
    which is a dictionary of pairs variable_name : value.
    Returns the request response, otherwise None.
    """
    try:
        with closing(requests.post(url,params)) as resp:
            if is_good_response(resp):
                return resp.content
```

```
            else:
                return None

    except RequestException as e:
        log_error('Error during requests to {0} : {1}'.format(url, str(e)))
        return None

def print_req(req):
    print('{}\n{}\n{}\n\n{}'.format(
        '-----------START-----------',
        req.method + ' ' + req.url,
        '\n'.join('{}: {}'.format(k, v) for k, v in req.headers.items()),
        req.body,
    ))

def read_params(path):
    params = {}
    with open(path,'r') as f:
        for line in f:
            if(len(line.split())==1):
                key = line.split()[0]
                val = None
            else:
                (key, val) = line.split()
            params[key] = val
    return params

print(simple_get("https://www.iberia.com/web/dispatchSearchFormHOME.do"))
```

Código 1: Funciones de POST con requests

Por no poder entender el funcionamiento de los POSTS de Iberia, nos vemos incapacitados para:

- Obtener el `.html` con la información del vuelo a buscar, en lugar de uno que nos informa del error y con información predeterminada.

- Saber manipular los parámetros de búsqueda como el que rellena el formulario de la página web.

Por ello, nos vemos obligados a abandonar la investigación con Iberia y cualquier otra aerolínea que requiera de POSTS siempre y cuando la dificultad de sus parámetros de entrada sea de este nivel. Abandonar Iberia sería en el caso de proseguir con `requests`, claro.

Ejemplo: GET Ryanair

Por otro lado, mostramos en esta subsección la simpleza de realizar un GET con Ryanair.

Si, por ejemplo, queremos buscar vuelos desde Madrid a Palma de Mallorca el 30 de Mayo de 2019, la URL del `GET` es:

`https://www.ryanair.com/es/es/booking/home/MAD/PMI/2019-05-30/1/0/0/0?Discount=0`

Es fácil ver que para poder elegir otros vuelos tan solo de cambiar los códigos de aeropuerto de origen (`MAD`) y destino (`PMI`), así como la fecha en formato `YYYY-MM-DD`. Los unos y ceros a la derecha de la fecha indican el número de distintos pasajeros para los que buscamos vuelo.

A continuación mostramos las funciones implementadas para `GET` de la URL dada:

```
def simple_get(url):
    """
    Attempts to get the content at `url` by making an HTTP GET request.
    If the content-type of response is some kind of HTML/XML, return the
    text content, otherwise return None.
    """
    try:
        with closing(get(url, stream=True)) as resp:
            if is_good_response(resp):
                return resp.content
            else:
```

```
            return None
    except RequestException as e:
        log_error('Error during requests to {0} : {1}'.format(url, str(e)))
        return None

url =
"https://www.ryanair.com/es/es/booking/home/MAD/PMI/2019-05-30/1/0/0/0?Discount=0"
print(simple_get(url))
```

Código 2: Funciones de GET con requests

No hay función alguna de modificar la URL según los destinos y la fecha (de momento) debido al hecho de que nos encontramos con un segundo y último problema al ejecutar este código.

Segundo Problema: Cabeceras y Scripts

El segundo problema de `requests` en nuestra investigación es de tal calibre que hace finiquitar este camino de abordaje al problema. Es decir, **no se van a sintetizar los procesos de búsqueda con requests**.

Dicho problema aparece cuando un host analiza las cabeceras de una petición de forma que rechace paquetes inadecuados. No especificar el navegador, fechas actuales, idioma, etc. son signos para dichos hosts de poder sufrir cualquier tipo de ataque (dado que es más probable que el tráfico sea sintético). Rechazar paquetes inadecuados puede traducirse en enviar `.html` de errores, y ello, por supuesto, no contiene información alguna sobre nuestras búsquedas.

Resulta que Ryanair parece aplicar este tipo de análisis en el tráfico que intercambia con nosotros y, con ello, nos responde con páginas predeterminadas. Podríamos intentar analizar qué headers son necesarios y cuáles no, pero hacer esto supone:

1. Alejarnos más de poder intercambiar tráfico canónico para poder manipularlo.
2. Demasiado tiempo analizando parámetros opacos sin la certeza de que vaya a funcionar.

Aparte, pese a que algunas veces hemos conseguido que el host nos aceptara, se añade

otra dificultad: **Un `GET` de scripts embebidos es sinónimo de no recibir los datos que queremos (puesto que dicho script no tiene navegador donde ejecutarse)**. En el caso de las aerolíneas, scripts que muestran resultados de búsqueda es el recurso más utilizado.

Para completar nuestra araña web basada en `requests`, solo faltaba parsear los `.html` obtenidos con una librería como `BeautifulSoup`. En `flight_search/requests` se adjuntan dos `.html`, uno de Expedia donde no se hace uso de estos filtros y, con ello, si parseamos con `grep`, por ejemplo, encontramos los precios de distintos vuelos. El otro `.html` es uno de Ryanair, con el cual vemos que no se obtiene ningún dato de precios buscados.

Síntesis

En definitiva, crear una araña web con `requests` es simple cuanto más simple sea el host de la página web. Para nuestra investigación, es claro que nos daría demasiadas dificultades llevar adelante la implementación con este paquete por estos motivos:

- Las peticiones POSTS que tengan parámetros difíciles de analizar serán imposibles de sintetizar a nuestro antojo.
- Cualquier petición en general hacia un host "exigente" con las cabeceras intercambiadas aumenta considerablemente la dificultad en conseguir generar nuestro propio tráfico.

Aparte, señalar que tenemos imposible desde `requests` poder comunicarnos desde IP "anónima" (o al menos variable para escapar del *IP tracking*), luego se está despreciando un factor importante para la discriminación perfecta.

5

Implementación: Selenium

Por supuesto, no todo está acabado tras ver que `requests` no es apto para responder a nuestra pregunta. Queda un último (último por el hecho de ser el que funcionó) recurso.

En `flight_search/selenium` está el programa `flight_search.py` (en un alarde de creatividad); que se comentará en este capítulo.

Pseudocódigo y Estructura

`flight_search` es un programa de búsquedas de vuelos hecho en Python dados una serie de argumentos de ejecución útiles para nuestra investigación (y para buscar vuelos como tal).

El pseudocógido de este programa es simple cuando no se entran en detalles de Selenium.

```
Recoger argumentos de entrada (verbosidad, privacidad y formulario de búsqueda)
Preparar driver de selenium según la privacidad dada
```

```
Según la compañía en la que buscar, entrar en su función de búsqueda
Recoger resultados de horas y precios
Si lo piden los argumentos:
  Guardar vuelo más barato con precio y timestamp en DB
Cerrar driver y finalizar.
```

Código 3: Pseudocódigo del flujo principal de flight_search

Este pseudocódigo es más bien una enumeración debido a su linealidad, pero considero importante tener claro este flujo principal dado que deja clara una modularización del código.

Con ello, a continuación se listan los módulos (y otros archivos) que dan lugar a `flight_search`:

- `cookie_data/`: Carpeta con las cookies que se cargan y almacenan en cada búsqueda realizada sin privacidad alguna.
- `drivers/`: Carpeta con los ejecutables de `geckodriver` y `PhantomJS`, necesarios para poder navegar automatizadamente con Selenium en Firefox y Tor, respectivamente.
- `flight_data/`: Carpeta que contiene:
  - `flight_data_db.sqlite`: Base de datos administrada con `sqlite3` con datos sobre vuelos buscados.
  - Múltiples `.txt` con datos sobre un vuelo en específico, con cada línea un par precio, timestamp de cuando se buscó.
- `flight_search.py`: Programa principal. Su código es únicamente el del pseudocódigo anterior con algunas funciones extra para guardar en la base de datos y ordenar según el precio.
- `Makefile`: Con instrucciones para lanzar los procesos de tandas de búsquedas.
- `log/`: Carpeta con logs de cuando se realiza una tanda de búsquedas. Es simplemente a donde paso la salida de ejecución de mi `Makefile` para después buscar qué errores han habido.
- `searches/`: Carpeta con bash scripts de las tandas de búsquedas de cada aerolínea y `.txt` con los vuelos que hemos fijado.
- `src/`: Carpeta con el código fuente del resto de módulos. Contiene:
  - `Driver.py`: Clase `Driver`, con funciones de preparación y cierre del Driver

de `selenium`.

- `Iberia.py` y `Ryanair.py`: Módulos que implementan la búsqueda de vuelos y recogida de precios y horas. Más adelante se explicará por qué hemos de distinguir compañías entre módulos.

Argumentos de Ejecución

Para entrar mejor en contexto con lo que le pediremos a `flight_search` de cara a nuestros experimentos, considero importante saber qué argumentos de ejecución tiene este programa. Los argumentos han sido recogidos mediante la librería `argparse` de Python.

La salida de `flight_search.py -h` nos informa bastante bien de dichos argumentos:

```
usage: flight_search.py [-h] [-c | -t] [-f] [-v] [-hd] company fromc to date

A headless flight searcher

positional arguments:
  company          select company of the flight
  fromc            from (recommended city name or airport code)
  to               to (recommended city name or airport code)
  date             date (MM/DD/YYYY)

optional arguments:
  -h, --help       show this help message and exit
  -c, --nocookies  disables cookies
  -t, --tor        search using TOR
  -f, --file       store (timestamp,depart hour,lowest price) in a generated
                   archive in 'flight_data/'
  -v, --verbosity  program verbosity
  -hd, --headless  use headless browser
```

Texto 4: Menú de ayuda de flight_search

Algunos comentarios sobre estos parámetros de entrada:

- **`-c` y `-t` son claves para nuestra investigación**:
  - Sin ninguno de ellos estamos en el escenario de buscar con cookies e IP tracking
  - Con `-c` nos situamos en navegación privada sin cookies pero con IP tracking.
  - Con `-t` nos situamos en el escenario de navegación privada, sin cookies y con IP "anónima" (consultar la sección 6 para entender estas comillas).
- `-f` es el encargado de "Si lo piden los argumentos" de nuestro pseudocódigo anterior.
- `-v` y `-hd` han sido utilizados para debugging. Con ellos, el programa iba informando de cada paso por el que iba, así como cuando no se introducía `-hd` se puede ver qué se va introduciendo en el navegador y cómo se avanza.

Selenium: Web Browser Automation

Antes de entrar en detalles de implementación una vez ya queda presentado `flight_search`, en esta sección daré a conocer a `selenium`.

Selenium es una librería implementada tanto para Python como para Javascript. A *grosso modo*, **su funcionamiento consiste en el de abrir un `driver` y ejecutar en dicho `driver` comandos como obtener URL's, buscar elementos HTML, rellenar campos en formularios, etc**. Por ello, Selenium es una librería bastante útil para el debugging de páginas web: ponemos a prueba la concurrencia del servidor y la robustez en cualesquiera parámetros de entrada para los formularios.

Por otro lado, claro está, esto convierte a Selenium en una herramienta para el lado opuesto de los usuarios de una página web: intentar buscar en ella vulnerabilidades, poner a prueba el servidor y, en el caso de nuestro proyecto, automatizar tareas de las que se debería encargar un humano. Dicha tarea es, por supuesto, buscar un vuelo y apuntar sus precios.

Se listan ahora algunas de las funciones principales para conseguir el funcionamiento de `flight_search`:

Inicialización del Driver

Para inicializar un Driver hemos de llamar al constructor adecuado en función del navegador que queramos utilizar. En nuestro caso, este podía ser o bien Firefox o bien Tor (más adelante se hablará de Selenium con Tor).

En el caso de Firefox, en una primera instancia el driver se asigna así:

```
driver = webdriver.Firefox()
```

Al `__init__()` de `Firefox` se le pueden asignar bastantes argumentos de entrada. De ellos destacamos `options` y `firefox_profile`. Ambos se inicializan con respectivos constructores y de estos cambiamos atributos del objeto. Por ejemplo, **para iniciar el driver de forma *headless* (no mostrar la ventana)**:

```
options = Options()
if headless and v:
  print("Setting headless mode...")
 options.headless = headless
```

Código 4: Estableciendo modo headless

De `firefox_profile` hablaremos en detalle en la sección Distinción de Contextos.

Cierre del Driver

Cerrar el driver inicializado es tan simple como:

```
driver.close()
```

Realmente en algunos contextos de distinción supondrá mayor esfuerzo cerrar el driver, pero para el caso general se procede de esta forma.

Cargar enlaces

Cargar un enlace también es fácil:

```
driver.get(url)
```

Y la distinción es mínima en este caso en función del contexto.

Encontrar Elementos e Interactuación

Pasamos a la sección clave para entender y manejar Selenium. Una vez ya tenemos nuestro driver con la configuración adecuada, en la URL objetivo, ya solo es cuestión de navegar "como un humano". Esto, en el lenguaje del paquete `selenium`, se traduce en **buscar elementos del HTML e interactuar**. Comentar que esto supone analizar una página con las herramientas de "Inspeccionar Elemento" y con ello saber cómo referenciárselos a nuestro driver.

Para buscar elementos tenemos varias opciones. Destacamos tres:

- `driver.find_elements_by_xpath(str)`: El método más recomendado. Xpath es sintaxis que define partes de un XML. Esto nos permite realizar *queries* bastante versátiles. Ejemplo:

`FromElement = driver.find_elements_by_xpath("//input[@placeholder='Departure airport']")` * `driver.find_elements_by_id(str)`: Este método y el siguiente pasan a estar incluidos dentro del primer método. Los he incluido dado que si está claro el campo que distingue nuestro elemento HTML de otros, podemos ahorrar sintaxis Xpath con estos otros métodos. El id de un elemento HTML es único por elemento, pero no todos los desarrolladores web se lo asignan a su página. Ejemplo:

`FromElement = driver.find_elements_by_xpath("//input[@placeholder='Departure airport']")` * `driver.find_elements_by_class(str)`: Su contexto de uso es el mismo que para `find_elements_by_id`. Ejemplo:

```
driver.find_elements_by_class_name('hour')
```

Un último aspecto importante a destacar a la hora de buscar elementos es que existen estos métodos exactamente iguales pero para encontrar uno único. Serían `find_element_by_xpath(str)` y análogos. Considero más correcto utilizarlos en plural por la comprobación de errores en la búsqueda y por la posibilidad de iterar con los resultados.

Pasamos ahora a las funciones de interactuación, que son métodos de la clase `WebElement`:

- `elm.clear()`: Usado en formularios. Vacía el contenido que haya en el campo de entrada.
- `elm.send_keys(str)`: Introduce `str` dentro del campo del formulario de `elm`.
- `elm.click()`: Usado con botones. Clica en el elemento web.

Los ejemplos de estos métodos se entenderán mejor en conjunto en la siguiente sección.

Ejemplo: Búsquedas con Iberia

A continuación mostramos un ejemplo simplificado para entender la idea de cómo hemos buscado vuelos en los compañías dadas. En una primera instancia Iberia dio menos problemas y era más simple, luego aquí está el ejemplo:

```
#Declaracion del driver
driver = webdriver.Firefox()

#Cargamos iberia.com
url='https://www.iberia.com/'
driver.get(url)

#Obtenemos el campo "desde" y "hacia"
FromElement = driver.find_elements_by_id('text-from-visible')
ToElement = driver.find_elements_by_id('text-to-visible')
if v:
    print('Inserting search parameters...')
if(FromElement and ToElement):
#Limpiamos los predeterminados (ubicación, busquedas anteriores)
#e introducimos los parámetros que introdujo el usuario
    FromElement[0].clear()
    ToElement[0].clear()
    #'\r\n' evitan que aparezcan pop-ups de destinos y autocomplete
    FromElement[0].send_keys(depart+'\r\n')
    ToElement[0].send_keys(arrive+'\r\n')
```

```
else:
    print('Fallo en la búsqueda de elementos')

#Las comprobaciones de encontrar el elemento están en cada búsqueda
#De ahora en adelante se omitirán por simplificar.

#Solo ida
NoReturnButton = driver.find_elements_by_id('ida')
NoReturnButton[0].click()

#Obtenemos el campo de Fecha y la introducimos
DateElement = driver.find_elements_by_id('diaSalida')
#Más adelante hablaremos de este sleep
sleep(1)
DateElement[0].send_keys(date+'\r\n\r\n')

#Y más adelante hablaremos de randomClick
randomClick = driver.find_elements_by_xpath('//button[@type="close"]')
randomClick[0].click()

#Obtenemos el boton de "Buscar"
SearchButton = driver.find_elements_by_id('toAvailSubmit')
#Y clicamos
SearchButton[0].click()
if v:
    print('Flight search queried, now waiting for results...')
sleep(Iberia.SEARCH_WAIT)
```

Código 5: Ejemplo de Iberia. Rellenando el formulario de búsqueda

Nota: Este código ya no es el de Iberia.py. Tuvo que actualizarse por motivos que se comentarán en el siguiente capítulo. Dado que el objetivo de este apartado es entender cómo interactuar con los nodos HTML y este ejemplo sigue siendo el más simple, no se actualiza.

De momento está la búsqueda realizada. Tras esperar un tiempo predefinido para

que carguen los resultados, se muestran los métodos usados para obtener los precios y horas de vuelo como parte de este ejemplo:

```
@staticmethod
def get_prices(driver,v=False):
    ret = []
    i = 1
    while True:
    #Es cuestión de inspeccionar la página de resultados de Iberia
    #para comprobar que se puede ir iterando según el nombre del id.
        PriceLabel =
          driver.find_elements_by_id('precio_ida-tarifa_1-vuelo_'+str(i))
        if not PriceLabel:
            break
        ret.append(PriceLabel[0].text)
        i+=1
    return ret

@staticmethod
def get_hours(driver,v=False):
    ret_list = []
    ret_tuples = []
    #Análogo a guardar precios pero iterando de cuatro en cuatro
    #para las horas.
    for element in driver.find_elements_by_class_name('hour'):
        ret_list.append(element.text.split('\n')[0])
    for i in range(0,len(ret_list),4):
        ret_tuples.append((ret_list[i],ret_list[i+1]))
    return ret_tuples
```

Código 6: Ejemplo de Iberia. Obteniendo precios y horas de vuelo.

Por último, `flight_search` mostrará los resultados encontrados, liberará todo recurso pedido y, si la flag `-f` estaba especificada, ordenará la lista de resultados por precio y se guardarán en un `.txt` y, más importante, en nuestra base de datos de búsquedas

de vuelos.

No considero importante mostrar en el ejemplo detalles sobre el uso de la librería `sqlite3` de Python, cómo ordenar los precios o cómo escribir en un fichero; puesto que se salen del objetivo de analizar y comprender el funcionamiento de Selenium.

Con el esqueleto de este ejemplo, **ya tenemos un autómata de búsquedas de vuelos**, pero esto es solo importante si no existieran (o no nos fiáramos) de los buscadores oficiales que ya existen en páginas web. A continuación se mostrará el *core* de este trabajo: **Las diferencias en código que consiguen distinguir unas privacidades de otras.**

Distinción de Contextos

Por último, pero no menos importante, presentaré los detalles de implementación que dan lugar a poder estar en un entorno de pruebas en el cual se puede decir que estamos diferenciando nuestras tres casuísticas principales.

Nuestras tres ramas de ejecución son similares en lo detallado en la sección anterior: Recoger parámetros de entrada, cargar la URL, rellenar el formulario y obtener los precios y horarios de vuelos. Las diferencias residen principalmente en el proceso de preparación del driver y detalles menores de interactuación con la web.

Con Cookies

Podríamos pensar en una primera instancia que la navegación con cookies mediante Selenium es trivial, pero lo cierto es que **Selenium inicia un nuevo perfil de navegador con cada instacia nueva del `driver`**. En cierto modo, esto equivale a como si instaláramos y desinstaláramos Firefox tras cada ejecución de `flight_search`.

Para solventar este contratiempo y simular un usuario normal sin privacidad, la solución está en guardar y cargar la configuración necesaria del navegador tras cada ejecución sin `-c` ni `-t`. Como lo que diferenciamos aquí es en el conservar IP y utilizar Cookies, lo anterior se traduce en hacer un `pickle` de nuestras cookies.

Antes de mostrar en código cómo cargar y guardar cookies, destacar algo más: **es necesario estar en la URL del dominio antes de cargar su cookie**. Esto puede suponer un problema, ¿y si desde el primer `GET` ya se nos está analizando lo que hacemos? Pese a que dicho primer `GET` no tenga tanta importancia como el de realizar búsquedas, nos curamos en salud de la siguiente manera: accediendo a una URL sin importancia del mismo dominio.

Una URL que no tiene importancia (a ojos de clientes usuales) común en todo dominio (para mayor generalidad del código) es `robots.txt`, un recurso ubicado al inicio de la mayoría de hosts web del que hablaremos en el capítulo 6.

Con ello, estas son las funciones de carga y descarga de cookies:

```
@staticmethod
def save_cookies(company,driver):
    pickle.dump(driver.get_cookies(),
                open("cookie_data/"+company+"_cookies.pkl","wb"))

@staticmethod
def load_cookies(company,driver,url):
    # este es el filename predeterminado segun la compañia
    file_name = "cookie_data/"+company+"_cookies.pkl"
    if os.path.exists(file_name):
        # vamos a robots.txt para ir al dominio
        def_url = url+'/robots.txt'
        driver.get(def_url)
        # cargamos el pickle
        cookies = pickle.load(open(file_name, "rb"))
        for cookie in cookies:
            try:
                # añadimos cada cookie de la lista guardada
                driver.add_cookie(cookie)
            except selenium.common.exceptions.InvalidCookieDomainException:
                try:
                    # para las cookies secundarias de otros dominios
                    driver.get(cookie['domain'])
```

```
                    driver.add_cookie(cookie)
                    driver.get(def_url)
                except:
                    print('Error: Invalid cookie:' + str(cookie))
                    cookies.remove(cookie)
    return driver
```

Código 7: Funciones de carga y descarga de cookies

Y, por ejemplo, las partes clave de carga y descarga de cookies en Ryanair:

```
#Al inicio de search

url='https://www.ryanair.com/'
if not disable_cookies and not tor:
    if v:
        print("Loading cookies...")
    driver = Driver.load_cookies('ryanair',driver,url)
    driver.get(url)

# [. . .]

#Tras clicar en botón de búsqueda

SearchButton[0].click()
if not disable_cookies:
    if v:
        print("Saving cookies...")
    Driver.save_cookies('ryanair',driver)

# [. . .]
```

Código 8: Ejemplo de carga y descarga de cookies

Sin Cookies, con IP Tracking

Después de lo explicado sobre Selenium y sus sesiones en el contexto de cookies e IP, se concluye fácilmente que no hemos de hacer nada para situarnos en este contexto cuando inicializamos `driver`. Pero, otra vez, para curarnos en salud, cambiamos la configuración de nuestro driver antes de acceder al correspondiente dominio.

Para ello, configuramos nuestro `driver` con un `firefox_profile` que navega en modo de incógnito (aunque sea redundante) y con las cookies desactivadas.

```
#En search de Driver.py
if disable_cookies:
  firefox_profile = webdriver.FirefoxProfile()
  firefox_profile.set_preference("browser.privatebrowsing.autostart", True)
  firefox_profile.set_preference("network.cookie.cookieBehavior",2)
  driver = webdriver.Firefox(options=options,firefox_profile=firefox_profile)
```

Código 9: Inicialización de un webdriver en incógnito con cookies deshabilitadas

Destacar que este último rasgo del `driver` genera diversos problemas en distintas webs:

- En Iberia aparece un *pop-up* informando sobre el uso de cookies y pidiendo que se activen (aunque realmente podemos navegar sin activarlas) que imposibilitaba interactuar con el resto de la web. Puede parecer de solución fácil, pero recordemos que hemos de interactuar con dicho *pop-up*, y precisamente el nodo HTML de este mensaje estaba programado para dificultar su cierre desde Selenium. Tras varios intentos, este es el código que ignora el *pop-up*, que hace uso de `ActionChains`.

```
if disable_cookies:
    # buscamos el boton de cierre del pop-up
    CookiesButtonElement = driver.find_elements_by_class_name('close')
    # creamos la cadena de acciones
    action = webdriver.common.action_chains.ActionChains(driver)
    # "movemos el puntero" (no altera el ratón) al botón de cerrar
    action.move_to_element_with_offset(CookiesButtonElement[0], 0, 0)
    # clicamos en el boton
```

```
        action.click()
        # pedimos que se performen las acciones especificadas
        action.perform()
```

Código 10: Cierre del pop-up sobre cookies

- En Ryanair directamente nos dejan en espera si detectan que tenemos las cookies desactivadas. Aparece una pantalla de carga (que nunca cargará). Pese a que parezca imposible realizar pruebas con este contexto, resulta que la página de Ryanair que contiene los resultados de vuelo no realiza este check. ¿Cómo acceder a dichos resultados si no podemos rellenar el formulario de búsqueda? La respuesta está en lo que llamo **"Atajos URL"**. Es básicamente hacer uso de lo aprendido en el capítulo anterior y llamar a `driver.get(url)`, con `url` una que varía en función de los parámetros de búsqueda. A continuación se muestra `shorcut` del paquete `Ryanair.py`:

```
@staticmethod
def shortcut(driver,display,depart,arrive,date,tor,v,headless):
    MM,DD,YYYY = date.split('/')
    uri_date = YYYY+'-'+MM+'-'+DD
    def_url = 'https://www.ryanair.com/'
    shortcut_url =
        'booking/home/'+depart+'/'+arrive+'/'+uri_date+'//1/0/0/0'
    if tor:
        driver.load_url(def_url)
        sleep(Ryanair.SEARCH_WAIT)
        url = driver.current_url + shortcut_url
        if v:
            print('Searching in '+ url + ' ...')
        driver.load_url(url)
        sleep(Ryanair.TOR_WAIT)
    else:
        url = def_url+'es/es/'+shortcut_url
        if v:
            print('Searching in '+ url + ' ...')
        driver.get(url)
```

```
        sleep(Ryanair.SEARCH_WAIT)
    if v:
        print('Getting results...')
    prices = Ryanair.get_prices(driver,v)
    if not prices:
        return None
    hours = Ryanair.get_hours(driver,v)

    return dict(zip(hours,prices))
```

Código 11: Implementación del "Atajo URL" de Ryanair

IP Anónima

Llegamos al contexto más difícil de este trabajo: IP anónima. Antes de continuar, comentar que es imposible (o al menos bastante difícil) ser IP anónima, y no es el caso de este contexto. Sin embargo, veremos que es posible escapar del IP tracking con el código que tenemos. La clave está en el uso de Tor, un navegador (más bien una rama de Firefox) con ciertas características especiales a la hora de navegar que lo hacen el mejor en términos de privacidad (excesiva para el objetivo de evitar IP tracking).

El primer rasgo característico de Tor es el que da origen a su icono (una cebolla): su infraestructura de red. El esquema que esta infraestructura sigue se puede ver como un grafo:

- Cada nodo del grafo es un servidor o usuario, que ofrece su equipo para ser utilizado en el proceso de conexión de Tor. A estos nodos se les suele llamar **relay**.
- De este grafo, con nuestro nodo como el inicial (que no quiere decir que nosotros ofrezcamos conexión a otros) trazamos un camino "aleatorio" de entre tres y cuatro nodos (por lo general). Este camino se conoce como **Tor bridge**.
- El nodo inicial del camino somos nosotros realizando peticiones HTTP al navegador, mientras que el nodo final es quien realmente se comunicará con el host web.
- Esta es la parte clave para comprender la seguridad de Tor: **cada salto de nodo**

en nodo supone encriptar en esquema híbrido todo el mensaje que vamos a enviar. Si el camino es de tres nodos, el procedimiento es así:

1. Establezco las clave entre Yo y R_1.
2. Le pido a R_1 que medie entre Yo y R_2, con quien establezco otra clave.
3. Ahora R_1 no sabe qué comunico con R_2, a quien le pido mediar con R_3.
4. Mis mensajes M serán $K_1(K_2(K_3(M)))$, con K_i la clave de sesión entre Yo y R_i (que ojo, iba encriptada en el intercambio de claves con la clave pública de R_i).

- Este esquema tiene las siguientes tres premisas importantes por el punto anterior:
  - R_i solo conoce a R_{i-1} y R_{i+1} (R_0 soy Yo y R_{N+1} es el host web).
  - R_1 es el único que conoce mi IP, pero no puede saber qué tráfico envío al seguir encriptado por las K_i con $i > 1$.
  - R_N es el único *relay* que descubre qué tráfico envío (M), pero no sabe quién ha sido al llevar el paquete la IP de R_{N-1}.

La realidad es que para evitar el IP tracking solo nos hace falta el hecho de que R_N variará al actualizar el *Tor bridge*, pero ya hemos visto que curarse en salud es un hábito en esta diferenciación de contextos.

Terminada la parte de teoría, pasamos al código, donde "dejamos de lado" `selenium` para pasar a `tbselenium`. Destacamos únicamente utilizar `stem` para inicializar el `driver` y el uso de `Xvfb` para ocultar el navegador en `flight_search -t --headless`:

```
#En prepare_driver de Driver.py

else:
  if v:
      print("Configuring tor browser...")
  tbb_dir = tor_path # este path se configurará en flight_search.conf
  if headless:
      xvfb_display = start_xvfb()
  try:
      tor_process = launch_tbb_tor_with_stem(tbb_path=tbb_dir)
```

```
    except OSError as e:
        if 'timeout' in str(e):
            print('Error: Tor connection timeout.' +
                  'Check URL or Internet connection')
            return None, None, None
        else:
            raise e
    driver = TorBrowserDriver(tbb_dir, tor_cfg=cm.USE_STEM)
    if headless:
        return driver, xvfb_display, tor_process
    else:
        return driver, None, tor_process
```

Código 12: Inicialización de un webdriver con Tor

La buena noticia es que **el resto del código es exactamente igual que el contexto de "Sin cookies"**, la mala es que en el siguiente capítulo tenemos bastantes comentarios sobre problemas que nos han dado los hosts web con Tor.

Síntesis

El final de este trabajo (y de este documento, al fin) se acerca. **Parecemos tener un entorno de pruebas adecuado para realizar mediciones**. Solo queda programar sencillos *bash scripts* que realizan lo que he llamado "lotes de búsquedas", que recogerán los resultados en una base de datos SQL (para extraer más fácilmente conclusiones, dado que serán bastantes los vuelos buscados).

Parecemos tener bastante evidencia y base cuantitativa a la conclusión que saquemos en el último capítulo pero, por supuesto, también es importante la calidad de dicha base. Todo el código del programa está a disposición del lector en caso de que este pueda encontrar una fuga importante (pese a mis múltiples vueltas a dicho código). Aparte, quizás puedan presentarse dualidades en opinión a ciertos criterios y consideraciones que he tomado en esta implementación.

Para dichas consideraciones, problemas que se antepusieron al proyecto en mitad de la

fase de recogida de datos e información adicional a tener en cuenta, está el penúltimo capítulo: “Previa a Conclusiones”

6

Previa a Conclusiones

Considero de tal vital importancia para la investigación y su veracidad los siguientes comentarios que he decidido incluirlos en un capítulo aparte previa al *finale*.

¿Qué esperamos encontrar?

Podemos presentar una dicotomía entre desenlaces de este trabajo, en base a lo encontrado en las estadísticas tomadas. El punto clave estará en analizar gráficas (una por vuelo investigado) de la siguiente forma:

- En el **eje X** tendremos el **tiempo**: una timestamp de cuándo se realizó la búsqueda.
- En el **eje Y** figura el **precio del billete de avión más barato** para el vuelo que estemos estudiando.
- Habrá tres gráficas en una:una línea de precios resultantes en búsquedas con cookies, otra en búsquedas sin cookies con IP tracking, y una última de búsquedas con Tor.

Con ello, dicha dicotomía se arbitra así en función de las gráficas:

- **NO hay discriminación de precios de primer grado según el uso de privacidad en búsquedas de vuelos por Internet:** Las tres líneas de cada método de búsqueda coinciden, o bien varían en mínimas fluctuaciones que incluso a veces desfavorezcan métodos óptimos según lo expuesto.
- **SÍ hay discriminación de precios de primer grado según el uso de privacidad en búsquedas de vuelos por Internet:** Las tres líneas de cada método no son coincidentes. O bien los métodos óptimos se van situando por debajo (más barato buscar, p.ej., con Tor que con cookies), o bien van coincidiendo hasta que el precio del vuelo más barato sube, momento en el que los métodos mejores permanecen "por debajo" un cierto margen de tiempo antes de subir para acompañar métodos en teoría peores.

Este último hecho tiene una limitación: En caso de estar pasando esto, dependemos de la velocidad en la toma de medidas de forma proporcional a cuánto margen existe. He realizado entre tres y diez lotes de búsquedas por día (en la conclusión mostraré cuántas búsquedas se realizaron). Quizás con un ordenador más potente (tener acceso a un cluster, p.ej.) o con mejor conexión a Internet habría sido posible mejorar la precisión en las medidas.

¿Discriminación de Precio?

Independientemente de los resultados finales, quería comentar este factor que puede ser importante para cualquiera que se identificó bastante con la puesta en contexto de la Introducción.

Los buscadores de vuelos no disponen del precio real siempre. Estos suelen actualizar todos sus precios entre dos y cuatro veces diarias. Disponer del precio actualizado cada segundo probablemente tenga la consecuencia de un gran gasto innecesarios de recursos.

Con ello, es posible que se tenga un vuelo fijado y que tengamos la mala suerte de entrar a comprar en ese intervalo de seis o doce horas en el cual subió el precio del billete. El hecho de que esta subida tenga lugar para todo el mundo a la vez es una de las materias que veremos en conclusiones, pero en la sección anterior se habló de la limitación existente a poder ver esto.

Por último, comentar que sí se ha demostrado que comprar varios billetes de forma seguida (pero no a la vez) supone aumentar el precio a dicha persona (para por ejemplo evitar la reventa de dichos billetes). La forma de comprobar que es "una misma persona" suele ser su IP, y esta es una razón crucial para aquellos que se identificaron en la Introducción cuando, por ejemplo, realizaron una compra de billetes entre varios amigos pero cada uno el suyo (puesto que probablemente estuvieran en la misma subred y, a ojos de los hosts web, fuera la misma IP pública).

El Vuelo Más Barato

Probablemente esta decisión de implementación afecte a la completitud en las medidas de este trabajo: **Siempre se guarda el vuelo más barato de los encontrados**. A continuación listo razones de la toma de esta decisión:

- Fluctuación de Precios: Son los vuelos más baratos los que más cambian de precio. Aquellos en hora punta o de clase *bussiness* apenas varían de precio. En mi opinión, esto ayudará además en conclusiones a obtener ideas de cuándo comprar un vuelo para obtenerlo más barato.
- Complejidad: Extraer todo vuelo de la página de resultados supone bastante dificultad en las funciones de obtención de precios (para un detalle que quizás en semanas pueda fallar, como explicará la sección siguiente). Aparte y más importante, considero que guardar todo vuelo podía complicar la extracción de conclusiones al estar realizando observaciones en tantos valores a variar a la vez.

Mantenimiento de `flight_search`

Incluyo esta sección aquí en caso de utilizarse `flight_search` más allá de mediados de 2019 (por si quiero usarlo para buscarme viajes de forma automática). Hay tres aspectos que dificultan la longevidad de nuestro programa, que suelen ocurrir en una coordinación de modularización unilateral (esto es, cuando mi programa depende de otro que desconoce mi existencia):

- Comprobación de conexión a Internet: Introduje un booleano en `flight_search.py` llamado `internet_on()`. Simplemente comprueba que hay respuesta en una

request que se hace a cierta IP (en este caso de Google). Destacar que se le pide a la IP y no al dominio por el hecho de que podríamos "quedarnos parados" en la resolución DNS en caso de no haber conexión, y esto es justo lo que estamos comprobando. A la mínima que deje de utilizarse la IP de Google, `flight_search` diagnosticará siempre que no hay conexión.

- URLs: Para acceder a cada compañía, es necesario requerir de su URL primero. A veces, usamos otras URLs cuando hay un "atajo" en la búsqueda, como es el caso de lo ya explicado en "Ejemplo: GET Ryanair" del capítulo 4. En el momento de cambiar una URL (más probable en el caso de los "atajos"), `flight_search` fallará por parsear en un *Not Found*.
- HTML de cada página: Este es el peor y más probable parámetro que dificulta el mantenimiento de `flight_search`. La gran desventaja de Selenium es que, si los desarrolladores Web del host sobre el que trabaja nuestra *spider* cambian `id`,`class`,`aria`, etc. de un elemento HTML, aquél parámetro mediante el cual buscábamos ha de adaptarse a las nuevas características del elemento. Lo más probable es que en algún momento cambie el diseño completo de páginas como Iberia o Ryanair, y ello conlleva rediseñar las funciones de completar formularios y obtención de precios y horas.

Legalidades de una Araña Web

Pese a que se tratara bastante qué podemos hacer investigando el funcionamiento de un sistema informático, faltaba un detalle importante a mencionar tras la implementación de `flight_search`. **Las arañas web tienen ciertos recursos del servidor web prohibidos**. Acceder a dichos recursos no tiene repercusiones penales (por lo general), tan solo supone un probable bloqueo de nuestra IP.

Los recursos a los que podemos acceder y a los que no figuran en `robots.txt`. Por ejemplo, para Iberia es `https://www.iberia.com/robots.txt`. En dicho archivo se encuentra una tabla de pares verbo-recurso donde el verbo es `allow` o `disallow` y el recurso una ruta (a veces aparece `*`, lo cual se interpreta como "todo archivo en").

Y aquí empieza una "batalla" entre *spiders* y *hosts web*, puesto que los *hosts* no suelen dar la bienvenida a ninguna araña salvo una: la del algoritmo *Page Rank* de Google. La cuestión es hasta dónde permitir acceso a dicha araña sin "ponernos en peligro" por

las demás.

Clicks y Esperas Aleatorias

Allá donde una araña puede no tener acceso según `robots.txt`, puede tener acceso un cliente normal y corriente. La siguiente parte de la "batalla" está en intentar **simular ser humano**. Esto se consigue por lo general mediante clicks en lugares aleatorios y en dormir (`sleep(segs)`) en momentos aleatorios. Cierto es que existen otras formas de descubrirnos como robots y no humano, como pueden ser las **honeytraps**: recursos HTML no visibles para un usuario pero que figuran en el `.html`, que al momento de interactuar con ellos "caemos en la trampa" y, por supuesto, es claro que el cliente es un *spider*.

Entonces entra la búsqueda del equilibrio entre aparentar ser humano para poder automatizar las tareas que deseamos y en la eficiencia de nuestra araña web (¿qué `sleeps` se salen de los necesarios y ralentizan el programa?). En nuestro caso veremos que tenemos ciertos matices de ello, pero en algunas compañías no es necesario porque surgirán problemas mayores.

El Problema con Iberia

La "guerra" entre hosts de servicios web y de clientes que se salen de lo habitual (como puede ser una araña web) da lugar a que, pese a dificultades y soluciones mencionadas en apartados anteriores, webs como Iberia tienen una medida fatídica contra `fligth_search`, tanto que me hizo decidir **apartar a Iberia del marco de investigación**.

Dicha solución es un *Captcha*. No creo que sea necesaria una introducción de lo que es. Si bien un *Captcha* tiene gran aplicación y sentido en contextos como ataques a contraseñas por fuerza bruta *online*, posteo de mensajes en un foro, etc., me pregunto los motivos por los que Iberia colocaría un Captcha entre la búsqueda de vuelos y obtención de resultados. Si alguien quiere poner a prueba (atacar) el servidor de Iberia, ¿no existen muchos otros métodos más eficientes que no requieren de rellenar formularios de forma automatizada?

Quizás la respuesta a la pregunta anterior tenga que ver con poder **sabotear sus estudios de mercado** (por ejemplo, con los recursos suficientes, simular que millones de clientes están interesados en volar de Madrid a Egipto).

Con ello, no era posible automatizar un lote de búsquedas, puesto que en menos de veinte búsquedas aparecía este *Captcha*, que se mantenía durante veinticuatro horas (y de hecho todo usuario de mi subred tenía que introducirlo).

Acceso a Iberia con Tor

`flight_search` **no funciona con Tor para Iberia**. El motivo creo que se escapa de mis capacidades. Iberia, directamente, bloquea toda conexión que sea mediante Tor, o al menos la mayoría de nodos desde los que he podido acceder (ha sido muy rara, un 3%, las ocasiones de prueba en las que conseguí conectarme desde Tor).

Este motivo, juntado con el de la sección anterior, fueron bastante importantes como para llevarme a cercar el marco de investigación únicamente en Ryanair.

Tor e IP Anónima

Contrario a lo que mucha gente pueda pensar al navegar con Tor mediante, **no tenemos IP anónima**. Distinto es el hecho de que dicha IP no sea nuestra, por lo ya explicado en *Distinción de Contextos: Tor*. Repetía este hecho por esta consideración en nuestra investigación: quizás se poluta nuestra medida que afirma no tener *IP tracking* por el hecho de que se aplique esta técnica a algunos nodos finales mediante los cuales hemos buscado.

Por supuesto, la probabilidad de esto es mucho menor, además porque apenas `flight_search -t` va repetir nodos. Considerado este detalle, ya es parte del lector opinar si la medida "Sin cookies e IP anónima" es verídica o no.

Tor y Determinismo en Fallos de Ejecución

`flight_search` **no siempre funciona con Tor**. Sin embargo, considero este un factor que se escapa de mi código. Por lo general, `flight_search -t` dependerá del último

nodo que conecta con el host web. Dichos nodos, en ocasiones, han sido utilizados "de forma maligna" (entre comillas al ser por lo general intentos fallidos de ataques), y ello conlleva un bloqueo (temporal o permanente) por parte de los hosts.

La única solución posible a este problema es el hecho de reintentar establecer conexión, reseteando el *Tor bridge* hasta un número máximo de veces que está establecido en la configuración de `flight_search` y refrescando el enlace.

Desafortunadamente, esta solución no consigue paliar del todo el problema y, en comparación con las búsquedas con y sin cookies, `flight_search -t` con Ryanair tiene una tasa de acierto de, aproximadamente, un 60%.

Se Pueden Conseguir Vuelos Más Baratos con Tor

Este aspecto es bastante importante para quien realmente quiera "ahorrar" comprando billetes de avión. **Tor puede realmente conseguir vuelos más baratos**. El truco está en el cambio de moneda. Hay varios artículos que demuestran que se puede ahorrar bastante por las fluctuaciones de diferencias monetarias en ciertas aerolíneas, y se debe a que estas en ocasiones adaptan el precio de un billete según ciertas monedas en lugar de realizar el cambio "automáticamente".

Para este trabajo este no es el objetivo: Deshabilité nodos finales de zonas que no fueran europeas. Además, para no poder errar en conclusiones por cambio de moneda, directamente decidí no implementar una `get_prices` que atendieran a algún precio que no fueran Euros.

7

Conclusiones

Finalizamos este trabajo mostrando los datos obtenidos más destacables y concluyendo sobre estos. Hablábamos en el capítulo anterior sobre la dicotomía de resultados ideales para poder concluir con un simple sí o un rotundo no y, como era de esperar, medir algo tan dinámico como los precios de avión supone encontrarnos con distintos grises que son mezcla del blanco y negro que esperábamos encontrar.

El inicio de la toma de datos tuvo lugar el **7 de Abril de 2019**. La toma de datos se dio por finalizada veinte días después, el **27 de Abril de 2019**.

Datos Generales

Previo a mostrar los datos obtenidos, damos números acerca de la base empírica de la que disponemos para estas conclusiones. Podemos "jugar" como queramos con los datos mediante la base de datos `flight_data_db.sqlite`, disponible en la carpeta `flight_search/selenium/flight_data`.

Total de Búsquedas Realizadas

- **Total de búsquedas realizadas:** 10314
- **Búsquedas realizadas con Cookies:** 3647 (35%)
- **Búsquedas realizadas sin Cookies en navegación privada:** 4296 (41%)
- **Búsquedas realizadas desde Tor:** 2371 (23%)
- **Búsquedas realizadas con Iberia:** 98

De ahora en adelante obviaremos las búsquedas que pudimos llegar a hacer con Iberia en este estudio. Los diversos motivos se argumentaron en el capítulo 6.

En una base empírica utópica no presentaríamos este desequilibrio entre búsquedas realizadas, en especial las búsquedas con Tor (pero ya comentamos por qué ocurriría esto en el capítulo anterior). Para "compensar", se fomentaron múltiples búsquedas redundantes sin cookies para "tener con quién competir".

Vuelos Fijados y Total de Búsquedas por Vuelo

A continuación mostramos una tabla con los vuelos que hemos fijado y el total de búsquedas de las que disponemos. Todo vuelo de ahora en adelante tiene como aeropuerto de salida Adolfo Suárez - Barajas (Madrid):

Aeropuerto llegada	Fecha de vuelo	Búsquedas realizadas
Aeropuerto de Bérgamo (BGY)	06-01-2019	136
Aeropuerto de Bérgamo (BGY)	06-02-2019	134
Aeropuerto de Bérgamo (BGY)	06-03-2019	135
Aeropuerto de Bérgamo (BGY)	06-04-2019	92
Aeropuerto de Bérgamo (BGY)	06-05-2019	120
Aeropuerto de Bérgamo (BGY)	06-06-2019	119
Aeropuerto de Bérgamo (BGY)	06-07-2019	125
Aeropuerto de Bérgamo (BGY)	06-08-2019	139
Aeropuerto de Bérgamo (BGY)	06-09-2019	131
Aeropuerto de Bérgamo (BGY)	06-10-2019	126
Budapest-Ferenc Liszt (BUD)	06-01-2019	158
Budapest-Ferenc Liszt (BUD)	06-02-2019	145

Aeropuerto llegada	Fecha de vuelo	Búsquedas realizadas
Budapest-Ferenc Liszt (BUD)	06-04-2019	125
Budapest-Ferenc Liszt (BUD)	06-05-2019	133
Budapest-Ferenc Liszt (BUD)	06-06-2019	106
Budapest-Ferenc Liszt (BUD)	06-08-2019	138
Budapest-Ferenc Liszt (BUD)	06-09-2019	139
Roma-Ciampino (CIA)	06-01-2019	146
Roma-Ciampino (CIA)	06-02-2019	129
Roma-Ciampino (CIA)	06-03-2019	131
Roma-Ciampino (CIA)	06-04-2019	128
Roma-Ciampino (CIA)	06-05-2019	134
Roma-Ciampino (CIA)	06-06-2019	135
Roma-Ciampino (CIA)	06-07-2019	137
Roma-Ciampino (CIA)	06-08-2019	132
Roma-Ciampino (CIA)	06-09-2019	129
Roma-Ciampino (CIA)	06-10-2019	139
Dublín (DUB)	06-01-2019	106
Dublín (DUB)	06-02-2019	102
Dublín (DUB)	06-03-2019	102
Dublín (DUB)	06-04-2019	68
Dublín (DUB)	06-05-2019	134
Dublín (DUB)	06-06-2019	116
Dublín (DUB)	06-07-2019	108
Dublín (DUB)	06-08-2019	101
Dublín (DUB)	06-09-2019	103
Dublín (DUB)	06-10-2019	100
Hamburgo (HAM)	06-02-2019	147
Hamburgo (HAM)	06-05-2019	133
Hamburgo (HAM)	06-07-2019	137
Hamburgo (HAM)	06-09-2019	142
Bucarest-Henry Coanda (OTP)	06-01-2019	137
Bucarest-Henry Coanda (OTP)	06-02-2019	142
Bucarest-Henry Coanda (OTP)	06-03-2019	153

Aeropuerto llegada	Fecha de vuelo	Búsquedas realizadas
Bucarest-Henry Coanda (OTP)	06-04-2019	141
Bucarest-Henry Coanda (OTP)	06-05-2019	146
Bucarest-Henry Coanda (OTP)	06-06-2019	149
Bucarest-Henry Coanda (OTP)	06-07-2019	149
Bucarest-Henry Coanda (OTP)	06-08-2019	148
Bucarest-Henry Coanda (OTP)	06-09-2019	140
Bucarest-Henry Coanda (OTP)	06-10-2019	139
London Stansted (STN)	06-01-2019	95
London Stansted (STN)	06-02-2019	102
London Stansted (STN)	06-03-2019	109
London Stansted (STN)	06-04-2019	76
London Stansted (STN)	06-05-2019	147
London Stansted (STN)	06-06-2019	143
London Stansted (STN)	06-07-2019	106
London Stansted (STN)	06-08-2019	103
London Stansted (STN)	06-09-2019	97
London Stansted (STN)	06-10-2019	104
Berlín-Schönefeld (SXF)	06-01-2019	123
Berlín-Schönefeld (SXF)	06-02-2019	122
Berlín-Schönefeld (SXF)	06-03-2019	128
Berlín-Schönefeld (SXF)	06-04-2019	125
Berlín-Schönefeld (SXF)	06-05-2019	124
Berlín-Schönefeld (SXF)	06-06-2019	129
Berlín-Schönefeld (SXF)	06-07-2019	129
Berlín-Schönefeld (SXF)	06-08-2019	133
Berlín-Schönefeld (SXF)	06-09-2019	125
Berlín-Schönefeld (SXF)	06-10-2019	127
Viena-Schwechat (VIE)	06-01-2019	131
Viena-Schwechat (VIE)	06-02-2019	127
Viena-Schwechat (VIE)	06-03-2019	130
Viena-Schwechat (VIE)	06-04-2019	125
Viena-Schwechat (VIE)	06-05-2019	106

Aeropuerto llegada	Fecha de vuelo	Búsquedas realizadas
Viena-Schwechat (VIE)	06-06-2019	126
Viena-Schwechat (VIE)	06-07-2019	137
Viena-Schwechat (VIE)	06-08-2019	133
Viena-Schwechat (VIE)	06-09-2019	136
Viena-Schwechat (VIE)	06-10-2019	133

Tabla 3. Total de búsquedas realizadas por vuelo.

Cada tanda se realizó de forma uniforme (todos los vuelos a buscar están en `search_ryanair.txt`, del cual leía el script `search_ryanair.sh`. Aquellas diferencias en número de búsquedas se deben o bien al indeterminismo de acierto con Tor o bien al mero hecho de llenarse el vuelo y no aparecer precios (al menos para clase turista).

Con estos datos, vemos que en estos veinte días de lotes se tomaron medidas cada entre tres y cuatro horas (realmente han habido lapsus de tiempo en los que me fue imposible tener el script activo).

Media de precios

- **Precio medio de vuelos buscados utilizando cookies:** 69.43 €
- **Precio medio de vuelos buscados sin cookies en navegación privada:** 68.91 €
- **Precio medio de vuelos buscados utilizando Tor:** 68.48 €

Este dato parece favorecer realizar búsquedas de vuelos con la mayor privacidad posible. Sin embargo, una diferencia de apenas un euro no creo que se le pueda llamar "diferencia considerable" que apoye un Sí.

Veamos esta media de precios en cada uno de los vuelos fijados:

Código de Aeropuerto	Fecha de vuelo	Precio medio Cookies (€)	Precio medio sin Cookies (€)	Precio medio Tor (€)
Aeropuerto de Bérgamo (BGY)	06-01-2019	45.45	45.69	45.2
Aeropuerto de Bérgamo (BGY)	06-02-2019	124.94	126.0	127.56
Aeropuerto de Bérgamo (BGY)	06-03-2019	64.54	64.36	61.91
Aeropuerto de Bérgamo (BGY)	06-04-2019	34.67	34.67	34.67
Aeropuerto de Bérgamo (BGY)	06-05-2019	25.95	26.51	26.66
Aeropuerto de Bérgamo (BGY)	06-06-2019	28.51	29.47	29.69
Aeropuerto de Bérgamo (BGY)	06-07-2019	34.34	34.31	33.54
Aeropuerto de Bérgamo (BGY)	06-08-2019	38.24	38.3	38.04
Aeropuerto de Bérgamo (BGY)	06-09-2019	39.23	39.09	38.97
Aeropuerto de Bérgamo (BGY)	06-10-2019	49.97	49.97	49.97

Código de Aeropuerto	Fecha de vuelo	Precio medio Cookies (€)	Precio medio sin Cookies (€)	Precio medio Tor (€)
Budapest-Ferenc Liszt (BUD)	06-01-2019	87.41	85.8	86.32
Budapest-Ferenc Liszt (BUD)	06-02-2019	52.57	53.77	55.13
Budapest-Ferenc Liszt (BUD)	06-04-2019	29.91	29.28	30.34
Budapest-Ferenc Liszt (BUD)	06-05-2019	30.79	30.51	30.96
Budapest-Ferenc Liszt (BUD)	06-06-2019	37.83	37.59	35.69
Budapest-Ferenc Liszt (BUD)	06-08-2019	208.74	210.83	216.31
Budapest-Ferenc Liszt (BUD)	06-09-2019	61.97	62.18	61.89
Roma-Ciampino (CIA)	06-01-2019	62.09	62.29	60.31
Roma-Ciampino (CIA)	06-02-2019	105.44	104.66	104.9
Roma-Ciampino (CIA)	06-03-2019	65.4	65.37	63.68

Código de Aeropuerto	Fecha de vuelo	Precio medio Cookies (€)	Precio medio sin Cookies (€)	Precio medio Tor (€)
Roma-Ciampino (CIA)	06-04-2019	48.8	49.03	48.61
Roma-Ciampino (CIA)	06-05-2019	53.99	53.92	53.32
Roma-Ciampino (CIA)	06-06-2019	52.34	52.06	50.58
Roma-Ciampino (CIA)	06-07-2019	74.93	73.1	71.04
Roma-Ciampino (CIA)	06-08-2019	64.43	64.72	64.93
Roma-Ciampino (CIA)	06-09-2019	42.85	42.1	40.48
Roma-Ciampino (CIA)	06-10-2019	53.24	53.07	52.43
Dublín (DUB)	06-01-2019	56.04	56.24	57.42
Dublín (DUB)	06-02-2019	154.24	149.64	145.07
Dublín (DUB)	06-03-2019	93.64	96.72	93.87
Dublín (DUB)	06-04-2019	46.15	45.89	50.6
Dublín (DUB)	06-05-2019	25.54	25.46	25.77

Código de Aeropuerto	Fecha de vuelo	Precio medio Cookies (€)	Precio medio sin Cookies (€)	Precio medio Tor (€)
Dublín (DUB)	06-06-2019	35.02	34.26	34.49
Dublín (DUB)	06-07-2019	44.28	44.77	46.67
Dublín (DUB)	06-08-2019	58.85	58.71	58.62
Dublín (DUB)	06-09-2019	64.83	64.98	63.97
Dublín (DUB)	06-10-2019	67.23	67.21	66.29
Hamburgo (HAM)	06-02-2019	117.33	117.4	116.63
Hamburgo (HAM)	06-05-2019	25.1	24.82	23.5
Hamburgo (HAM)	06-07-2019	31.48	31.39	31.09
Hamburgo (HAM)	06-09-2019	29.45	29.52	29.84
Bucarest-Henry Coanda (OTP)	06-01-2019	76.69	77.81	74.25
Bucarest-Henry Coanda (OTP)	06-02-2019	113.27	113.57	115.7
Bucarest-Henry Coanda (OTP)	06-03-2019	65.34	65.34	66.24

Código de Aeropuerto	Fecha de vuelo	Precio medio Cookies (€)	Precio medio sin Cookies (€)	Precio medio Tor (€)
Bucarest-Henry Coanda (OTP)	06-04-2019	59.42	58.83	59.07
Bucarest-Henry Coanda (OTP)	06-05-2019	79.28	78.6	79.75
Bucarest-Henry Coanda (OTP)	06-06-2019	61.09	61.19	60.91
Bucarest-Henry Coanda (OTP)	06-07-2019	77.6	77.6	76.57
Bucarest-Henry Coanda (OTP)	06-08-2019	67.6	68.17	68.05
Bucarest-Henry Coanda (OTP)	06-09-2019	77.8	78.67	79.27
Bucarest-Henry Coanda (OTP)	06-10-2019	67.15	67.1	67.35
London Stansted (STN)	06-01-2019	77.58	69.54	70.28

Código de Aeropuerto	Fecha de vuelo	Precio medio Cookies (€)	Precio medio sin Cookies (€)	Precio medio Tor (€)
London Stansted (STN)	06-02-2019	261.84	262.18	262.13
London Stansted (STN)	06-03-2019	154.01	148.79	146.4
London Stansted (STN)	06-04-2019	36.74	36.07	34.96
London Stansted (STN)	06-05-2019	24.32	18.92	19.43
London Stansted (STN)	06-06-2019	18.23	18.16	18.23
London Stansted (STN)	06-07-2019	37.43	24.01	25.78
London Stansted (STN)	06-08-2019	31.58	31.73	31.11
London Stansted (STN)	06-09-2019	36.65	36.66	36.5
London Stansted (STN)	06-10-2019	59.55	59.66	59.68
Berlín-Schönefeld (SXF)	06-01-2019	176.67	175.49	174.75

Código de Aeropuerto	Fecha de vuelo	Precio medio Cookies (€)	Precio medio sin Cookies (€)	Precio medio Tor (€)
Berlín-Schönefeld (SXF)	06-02-2019	138.83	136.11	136.12
Berlín-Schönefeld (SXF)	06-03-2019	84.78	84.77	87.14
Berlín-Schönefeld (SXF)	06-04-2019	98.91	97.18	92.81
Berlín-Schönefeld (SXF)	06-05-2019	20.14	20.22	20.28
Berlín-Schönefeld (SXF)	06-06-2019	20.13	20.22	20.27
Berlín-Schönefeld (SXF)	06-07-2019	108.45	107.96	106.76
Berlín-Schönefeld (SXF)	06-08-2019	195.2	194.53	188.48
Berlín-Schönefeld (SXF)	06-09-2019	58.11	57.51	57.96
Berlín-Schönefeld (SXF)	06-10-2019	177.54	179.59	179.98
Viena-Schwechat (VIE)	06-01-2019	63.36	62.98	62.21

Código de Aeropuerto	Fecha de vuelo	Precio medio Cookies (€)	Precio medio sin Cookies (€)	Precio medio Tor (€)
Viena-Schwechat (VIE)	06-02-2019	168.16	168.86	171.35
Viena-Schwechat (VIE)	06-03-2019	61.01	60.1	59.41
Viena-Schwechat (VIE)	06-04-2019	29.33	29.46	27.97
Viena-Schwechat (VIE)	06-05-2019	26.2	25.3	24.74
Viena-Schwechat (VIE)	06-06-2019	25.77	25.39	24.78
Viena-Schwechat (VIE)	06-07-2019	37.78	37.17	35.55
Viena-Schwechat (VIE)	06-08-2019	35.46	35.43	36.09
Viena-Schwechat (VIE)	06-09-2019	56.04	56.23	53.42
Viena-Schwechat (VIE)	06-10-2019	86.24	85.96	85.53

Tabla 4. Medias de precios en cada vuelo según el tipo de búsqueda.

Las medias mostradas pueden haberse visto polutadas por el mero hecho de aquellos lotes de búsqueda donde alguno de los tres tipos fallara. La tabla es icónica para ver

que, pese al comentario anterior, sí podemos afirmar que en un 80% de los vuelos realizados ha sido más barata la búsqueda con Tor.

Sin embargo, estas diferencias de no más de diez euros apenas tienen suficiente importancia (mucho menos cuando no llega ni al euro la diferencia). Con ello, **esta tabla tampoco tiene peso como para acompañar al Sí**.

Cuenteo de Discrepancias

Hemos fijado el calibre de tiempo de diferencia en treinta segundos. Es decir, nos hemos fijado en las comparaciones de mismos vuelos de distintos métodos de búsquedas donde el campo `ts` difiere en menos de treinta segundos. De esta forma retiramos bastante probabilidad a haber discrepancia por "haber subido el precio en ese momento". La excepción es `nocookies - tor`, que aumentamos a un minuto puesto que resultan ser el método más rápido y más lento de encontrar vuelo, respectivamente. Por ello, en el calibre de treinta segundos apenas llegaban a coincidir.

- **Número total de comparaciones `cookies - nocookies`:** 3558
- **Número total de comparaciones `cookies - tor`:** 1320
- **Número total de comparaciones `nocookies - tor`:** 1907
- **Número total de discrepancias en precio `cookies - nocookies`**: 29
- **Número total de discrepancias en precio `cookies - tor`**: 12
- **Número total de discrepancias en precio `nocookies - tor`**: 18

Aeropuerto	Fecha de vuelo	Diferencia de tiempo (s)	Discrepancia en Precio (cookies-nocookies) (€)
London Stansted (STN)	06-05-2019	22.0776240825653	296.2
London Stansted (STN)	06-07-2019	21.4069156646729	283.56
London Stansted (STN)	06-07-2019	22.9809563159943	240.1
London Stansted (STN)	06-01-2019	19.1717040538788	237.66

Aeropuerto	Fecha de vuelo	Diferencia de tiempo (s)	Discrepancia en Precio (cookies-nocookies) (€)
London Stansted (STN)	06-03-2019	24.2159004211426	182.58
Viena-Schwechat (VIE)	06-02-2019	20.6804165840149	22.79
London Stansted (STN)	06-03-2019	20.5641567707062	16.22
Berlín-Schönefeld (SXF)	06-04-2019	16.300856590271	14.59
Bucarest-Henry Coanda (OTP)	06-07-2019	2.98735046386719	9.85
Bucarest-Henry Coanda (OTP)	06-03-2019	19.1109838485718	8.62
Bucarest-Henry Coanda (OTP)	06-04-2019	21.4516522884369	7.29
Viena-Schwechat (VIE)	06-03-2019	20.3699281215668	6.53
Viena-Schwechat (VIE)	06-01-2019	22.1438539028168	6.53
Roma-Ciampino (CIA)	06-04-2019	4.69453716278076	5.21
Hamburgo (HAM)	06-07-2019	18.820365190506	4.48
Roma-Ciampino (CIA)	06-05-2019	18.5847783088684	1.02
Roma-Ciampino (CIA)	06-05-2019	20.5767154693604	-1.02
Aeropuerto de Bérgamo (BGY)	06-09-2019	19.3139600753784	-4.21
Viena-Schwechat (VIE)	06-07-2019	1.2151460647583	-4.77

Aeropuerto	Fecha de vuelo	Diferencia de tiempo (s)	Discrepancia en Precio (cookies-nocookies) (€)
Roma-Ciampino (CIA)	06-04-2019	21.4966557025909	-6.63
London Stansted (STN)	06-10-2019	21.1324706077576	-7.29
Budapest-Ferenc Liszt (BUD)	06-09-2019	16.2978050708771	-7.68
Roma-Ciampino (CIA)	06-03-2019	17.18306016922	-7.96
Roma-Ciampino (CIA)	06-06-2019	20.7256619930267	-7.96
London Stansted (STN)	06-10-2019	4.88407874107361	-8.05
Bucarest-Henry Coanda (OTP)	06-08-2019	25.1841547489166	-8.62
Roma-Ciampino (CIA)	06-01-2019	20.2241334915161	-9.55
Hamburgo (HAM)	06-02-2019	18.81232213974	-12.87
Viena-Schwechat (VIE)	06-02-2019	21.8137505054474	-20.82
London Stansted (STN)	06-02-2019	22.0281670093536	-34.61

Tabla 5. Discrepancias entre "Cookies" y "No Cookies"

Aeropuerto	Fecha de vuelo	Diferencia de tiempo (s)	Discrepancia en Precio (cookies-tor) (€)
London Stansted (STN)	06-05-2019	15.016	296.2
London Stansted (STN)	06-07-2019	13.267	283.56

Aeropuerto	Fecha de vuelo	Diferencia de tiempo (s)	Discrepancia en Precio (cookies-tor) (€)
London Stansted (STN)	06-03-2019	21.040	182.58
Berlín-Schönefeld (SXF)	06-04-2019	22.165	14.59
Bucarest-Henry Coanda (OTP)	06-07-2019	24.077	9.85
Bucarest-Henry Coanda (OTP)	06-04-2019	21.390	7.29
Hamburgo (HAM)	06-07-2019	23.307	4.48
London Stansted (STN)	06-06-2019	0.134	2.43
Dublín (DUB)	06-05-2019	23.914	-3.24
Viena-Schwechat (VIE)	06-09-2019	16.943	-5.2
Viena-Schwechat (VIE)	06-08-2019	22.830	-5.38
Bucarest-Henry Coanda (OTP)	06-01-2019	26.135	-9.85
Dublín (DUB)	06-03-2019	17.913	-13.13

Tabla 6. Discrepancias entre "Cookies" y "Tor"

Aeropuerto	Fecha de vuelo	Diferencia de tiempo (s)	Discrepancia en Precio (nocookies-tor) (€)
Dublín (DUB)	06-03-2019	40.0769531726837	13.13
Viena-Schwechat (VIE)	06-10-2019	39.5349283218384	11.39
Bucarest-Henry Coanda (OTP)	06-01-2019	29.3523256778717	9.84

Aeropuerto	Fecha de vuelo	Diferencia de tiempo (s)	Discrepancia en Precio (nocookies-tor) (€)
Viena-Schwechat (VIE)	06-08-2019	41.0935711860657	5.38
Viena-Schwechat (VIE)	06-09-2019	34.7482438087463	5.2
Aeropuerto de Bérgamo (BGY)	06-07-2019	52.8783011436462	3.58
Dublín (DUB)	06-05-2019	40.7062463760376	3.24
London Stansted (STN)	06-06-2019	49.4031088352203	2.26
Roma-Ciampino (CIA)	06-05-2019	33.8436281681061	1.02
Viena-Schwechat (VIE)	06-05-2019	57.7968423366547	0.97
Hamburgo (HAM)	06-09-2019	36.5994591712952	-3.85
Aeropuerto de Bérgamo (BGY)	06-09-2019	50.3580946922302	-4.21
Viena-Schwechat (VIE)	06-07-2019	21.1701905727386	-4.77
Roma-Ciampino (CIA)	06-04-2019	36.2511336803436	-6.63
bud	06-09-2019	43.2106504440308	-7.68
London Stansted (STN)	06-10-2019	31.1025021076202	-8.05
Roma-Ciampino (CIA)	06-01-2019	47.011670589447	-9.55
Aeropuerto de Bérgamo (BGY)	06-02-2019	38.3528325557709	-21.18
London Stansted (STN)	06-02-2019	32.0667531490326	-34.61

Tabla 7. Discrepancias entre "No Cookies" y "Tor"

Vemos que es un suceso bastante inusual el de encontrar una discrepancia, de aproximadamente un **0.86% de probabilidades**. Pese a dicha probabilidad, debemos tener bastante en cuenta las diferencias en precio de algunas de ellas, en especial destacamos entre "Cookies" y los otros dos métodos Stansted (Londres), con una **diferencia de precio de casi 300 euros**. Más adelante intentaremos dar una hipótesis a esta abismal diferencia cuando mostremos sus gráficas.

Por último, pasemos a ver las gráficas de tiempo/precio para los vuelos fijados en la investigación.

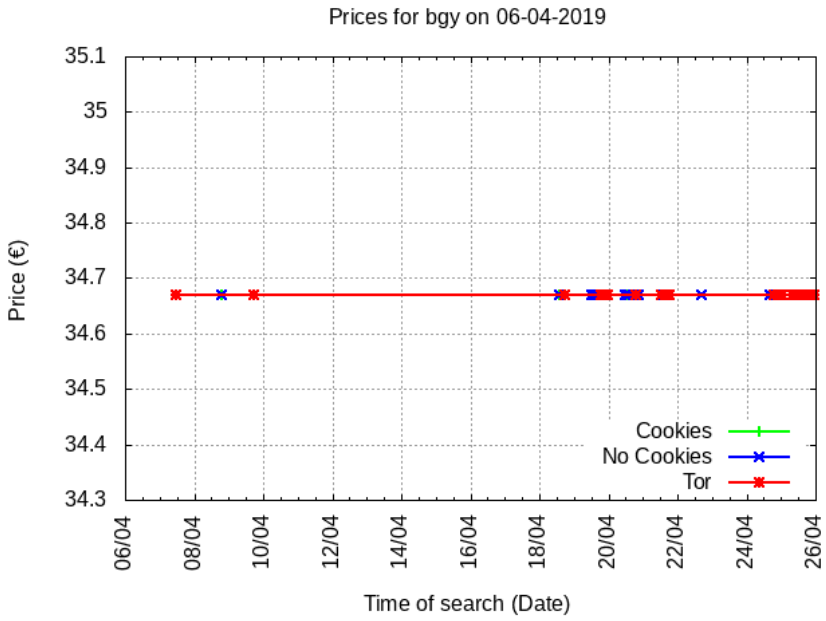
Gráficas a Destacar

No vamos a incorporar todas las gráficas a la memoria del trabajo: ello supondría aumentar en unas 60 páginas (cuando este trabajo ya es lo suficientemente extenso) con un mínimo de tamaño para la apreciación.

Con ello, vamos a incluir en esta última sección aquellas gráficas más distintivas, que puedan recoger a otras análogas y que tengan ciertos comentarios importantes a destacar.

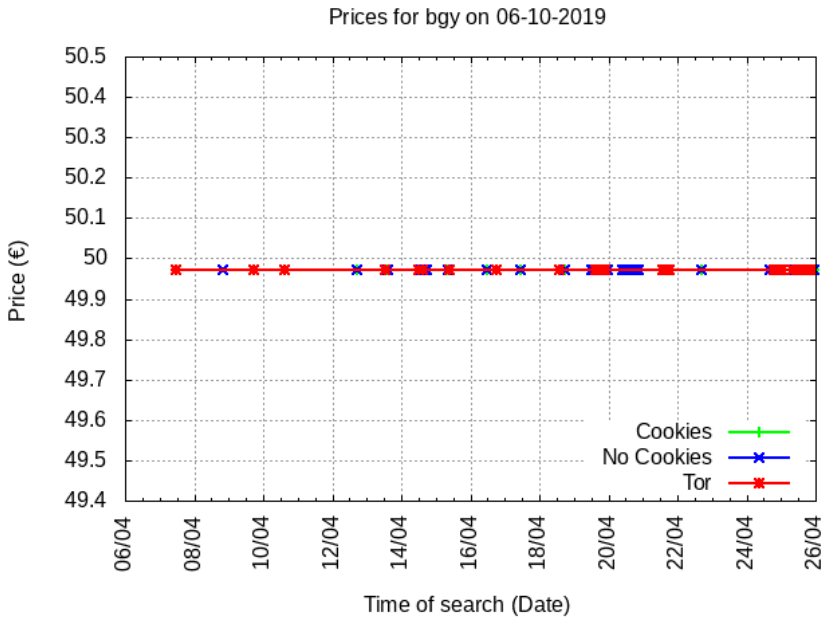
Caso 1: Precio Invariante

BGY:04/06 y BGY:10/06 son dos ejemplos de vuelos donde en todo momento el precio ha sido coincidente y único:



Las conclusiones con esta gráfica son simples: Tenemos vuelos que, debido a ser tan comunes y usuales, tienen un precio fijo invariante. El hecho de mostrar interés en estos vuelos parece serle indiferente a toda empresa de marketing y data analysis.

Con ello, en estos contextos, **no se ha experimentado discriminación de primer grado**.



Caso 2: Fluctuaciones

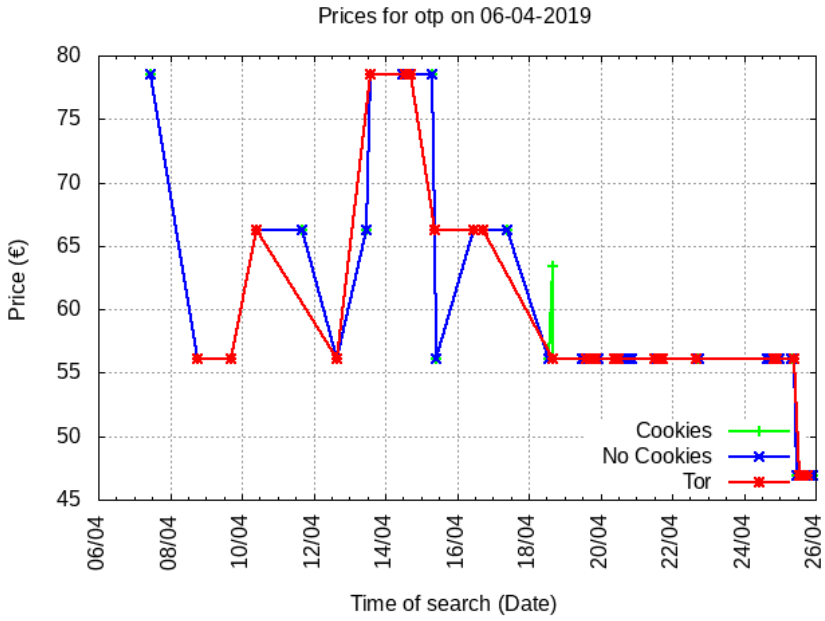
OTP:05/06 y la mayoría de los de este aeropuerto son el ejemplo del caoticismo de los precios de un vuelo:

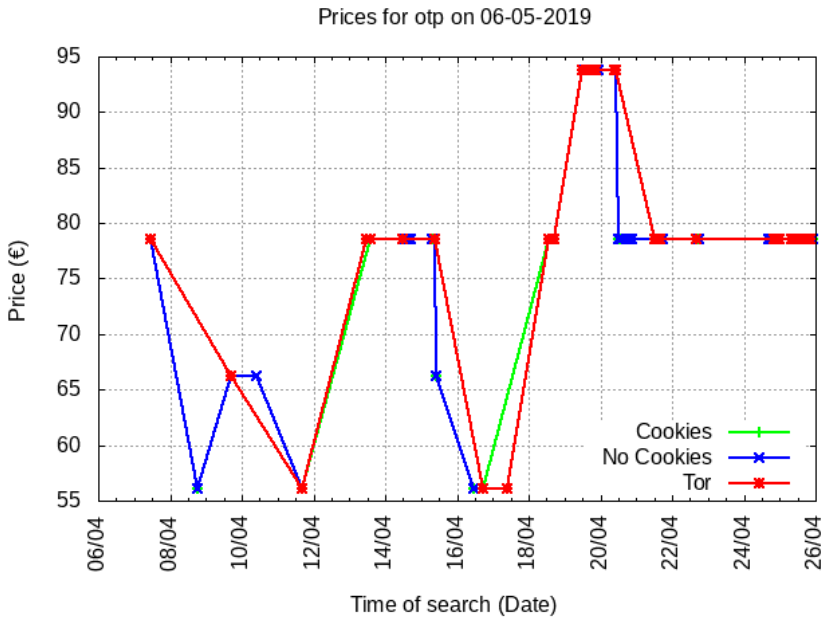
Ahora se complica cualquier conclusión a sonsacar.

En primer lugar, recordar que es distinto el número de búsquedas realizadas según el tipo, y no hemos de confundirnos si vemos alguna de las tres gráficas por encima o por debajo: Si no hay punto en la parte que discierne del resto, quiere decir que falta una medida, y es por ello que no se ha registrado dicho precio en búsqueda alguna.

Por otro lado, destacar que en este tipo de gráficas a veces salen perjudicadas las búsquedas con mayor privacidad y otras veces la búsqueda con cookies (recordemos, en color verde). Hay incluso una tanda de búsquedas del 15 de Abril donde, fijándonos en OTP:04/06, se llega a posicionar casi a la vez tanto por encima como por debajo el precio de buscar sin cookies y con cookies; frente a la gráfica de precios de Tor.

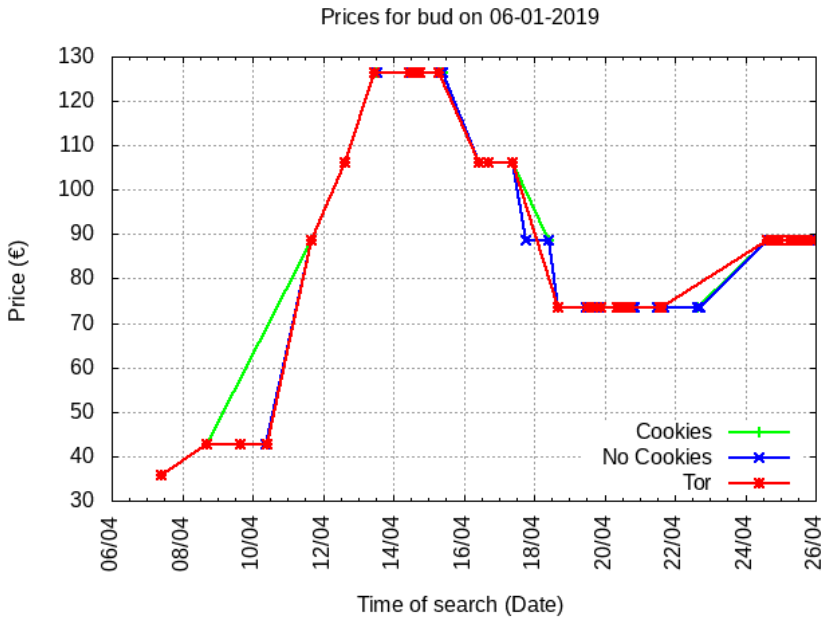
Por último, destacar un pequeño "pico" que ocurre el 19 de Abril para OTP del 4 de Abril. Este tipo de picos tienen cabida en una sección importante posterior.


Prices for otp on 06-04-2019
Price (€)
80
75
70
65
60
55
50
45
06/04
08/04
10/04
12/04
14/04
16/04
18/04
20/04
22/04
24/04
26/04
Time of search (Date)
Cookies
No Cookies
Tor

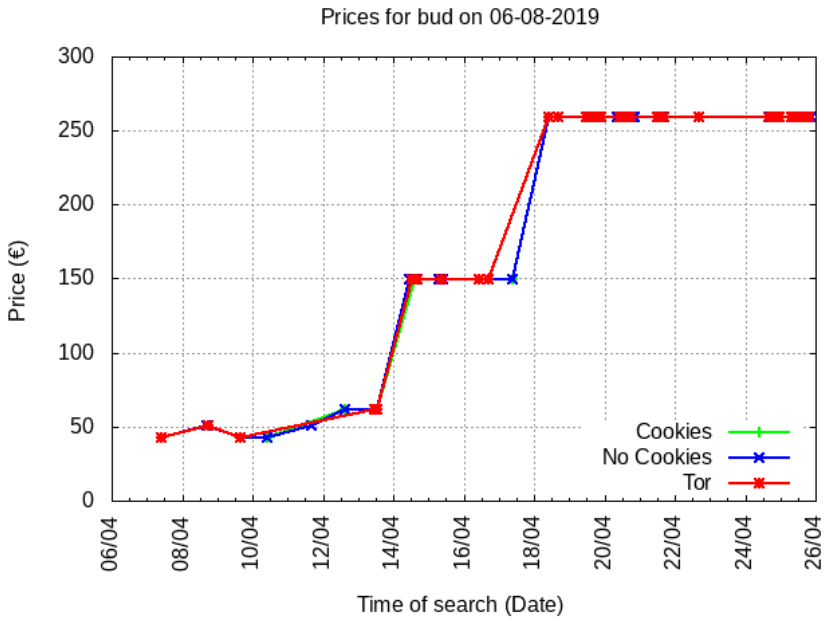

Prices for otp on 06-05-2019
Price (€)
95
90
85
80
75
70
65
60
55
06/04
08/04
10/04
12/04
14/04
16/04
18/04
20/04
22/04
24/04
26/04
Time of search (Date)
Cookies
No Cookies
Tor

Caso 3: Bajada o Subida Estable

BUD:01/06 y BUD:08/06 son claros ejemplos de cuando el precio de un vuelo puede bajar o subir pero los tres métodos de búsqueda coinciden al unísono. Esta coincidencia tiene lugar salvo, por supuesto, cuando es alguna búsqueda de algún tipo la que ha fallado (recordemos prestar atención a los puntos):



Desde mi criterio considero estos ejemplos otros claros casos donde podemos responder a nuestra pregunta con un **no, no parece variar el precio de un vuelo según la privacidad usada al buscar**.



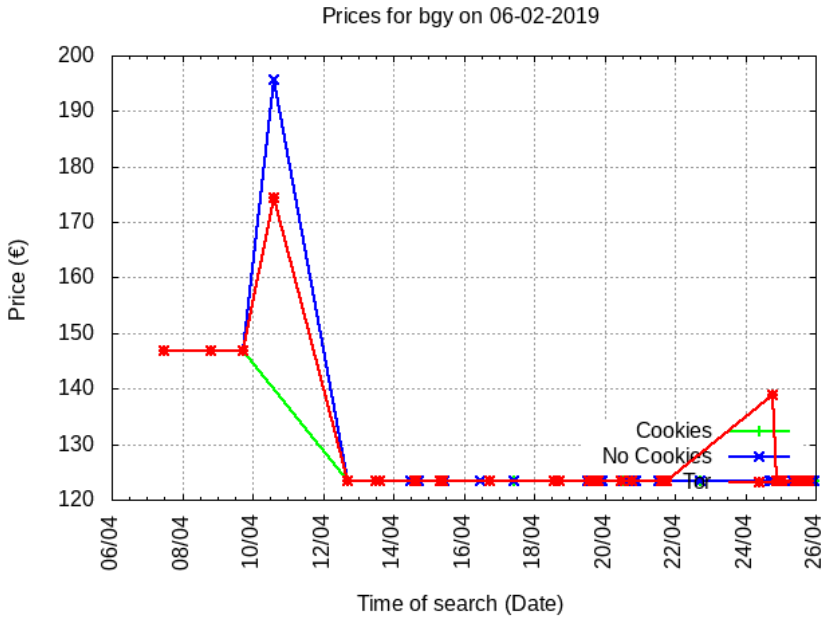
Caso 4: Picos Puntuales que no dependen del tipo de búsqueda

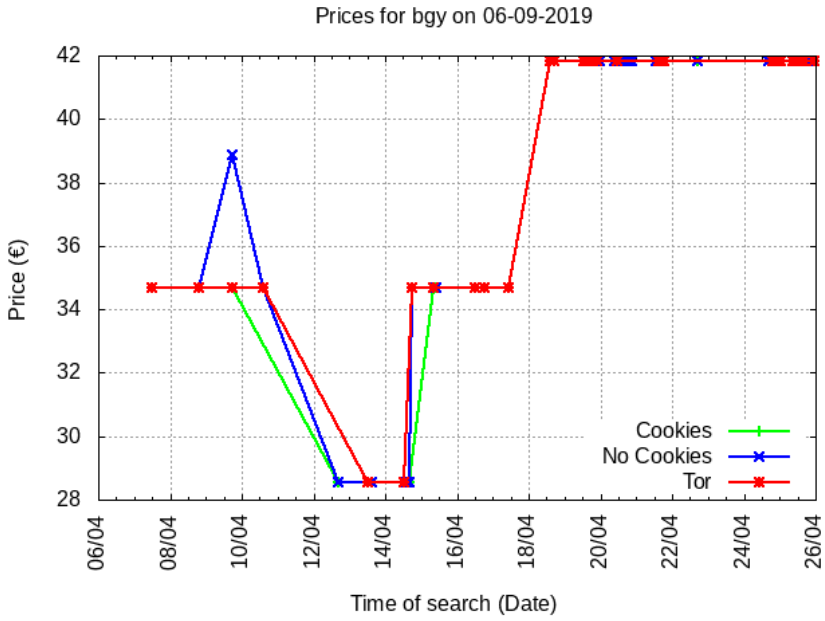
Presentamos cuatro vuelos en cuyas búsquedas hemos encontrado discrepancias entre tipos de búsqueda. Estas gráficas se tratan de ligeras curvas de precio a lo largo de estos veinte días de mediciones que, por otro lado, presentan "picos" en el precio donde algunos de los métodos de búsqueda pasan a diferenciarse.

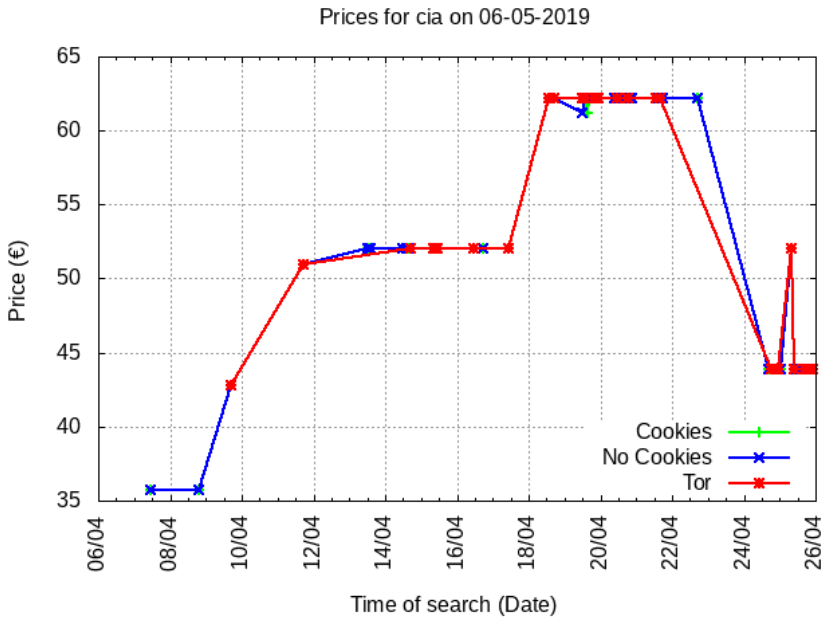
Esto sería positivo para la respuesta del sí, si no fuera porque **dichos picos nunca se decantan por un tipo en particular**. En BGY:02/06 el 11 de Abril Tor "sale ganando" frente a buscar sin Cookies: El IP tracking parece costarnos unos veinte euros de diferencia a quienes se conforman con modo de incógnito. En cambio, en este mismo vuelo, el 25 de Abril pasan los precios de vuelos a "penalizar" a los usuarios de Tor con una diferencia de unos quince euros.

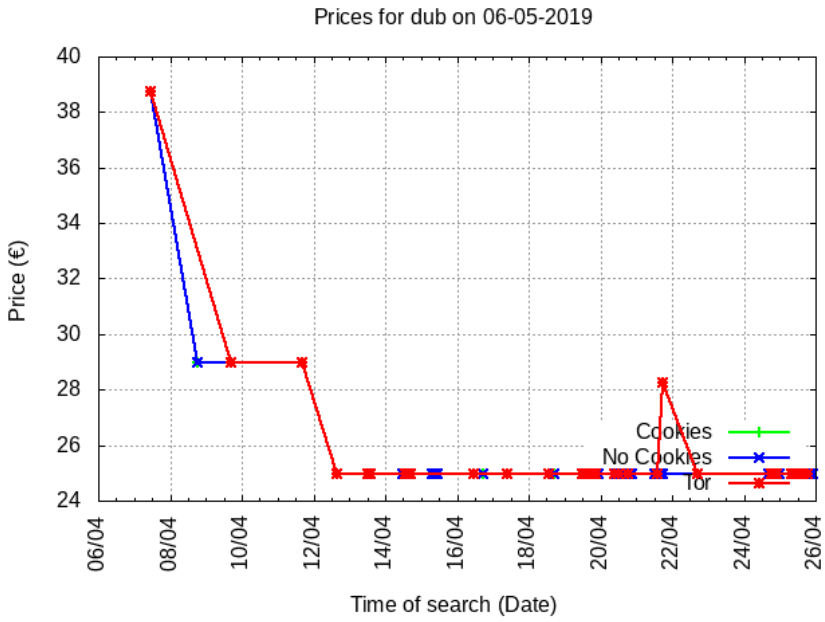
En las demás gráficas vemos que ocurre parecido: A veces Tor está por encima de los otros dos métodos, otras veces `nocookies` es el "autor" de dicho pico. Y lo que parece ser la verdad es que quizás no importa el método con el que buscamos (al menos en contextos de este estilo): **A veces uno de los tres sale ganando o perdiendo, pero ante todo la búsqueda con Cookies está coincidiendo con Tor y con la navegación**

privada. Es elección del usuario qué metodo elegir y tener la suerte de no encontrarse con alguna de estas numerosas fluctuaciones.




Prices for bgy on 06-09-2019
Price (€)
Time of search (Date)
Cookies
No Cookies
Tor
42
40
38
36
34
32
30
28
06/04
08/04
10/04
12/04
14/04
16/04
18/04
20/04
22/04
24/04
26/04


Prices for cia on 06-05-2019
Price (€)
Time of search (Date)
Cookies
No Cookies
Tor
65
60
55
50
45
40
35
06/04
08/04
10/04
12/04
14/04
16/04
18/04
20/04
22/04
24/04
26/04



Caso 5: Curiosos Picos en el Método con Cookies

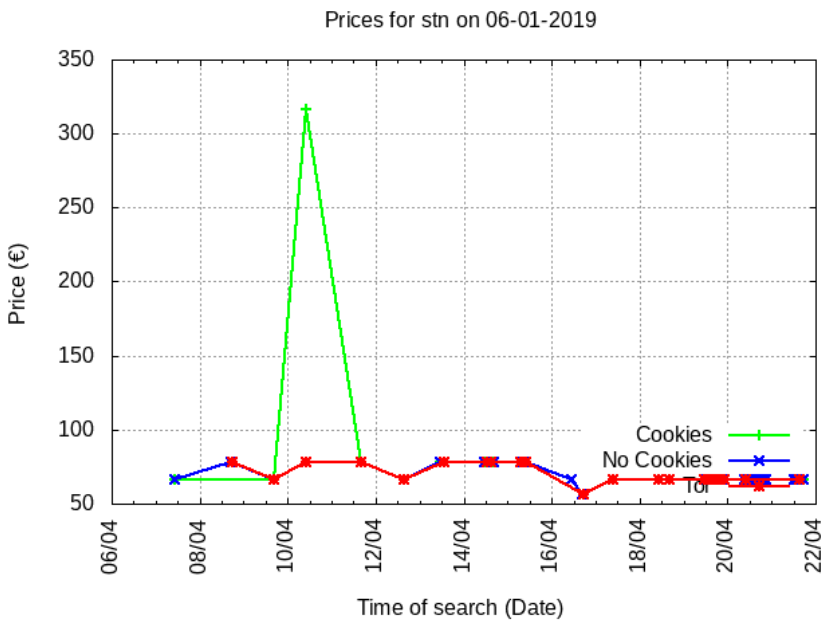
Me reservo la última casuística para la que más ha llamado mi atención. A continuación se muestran gráficas donde encontramos diferencias significativas únicas para el método de búsqueda con cero privacidad.

Encontramos diferencias de entre 100 y 300 euros únicas para la búsqueda con cero privacidad. Es completamente desconocida la causa de dichas subidas repentinas y tan significativas. Si para el contexto anterior dichos "picos" podían deberse a variar los asientos del vuelo más barato (de donde sale la pregunta de por qué según qué método varía el número de asientos), en ninguno de ellos llegábamos a experimentar una subida puntual tan significativa.

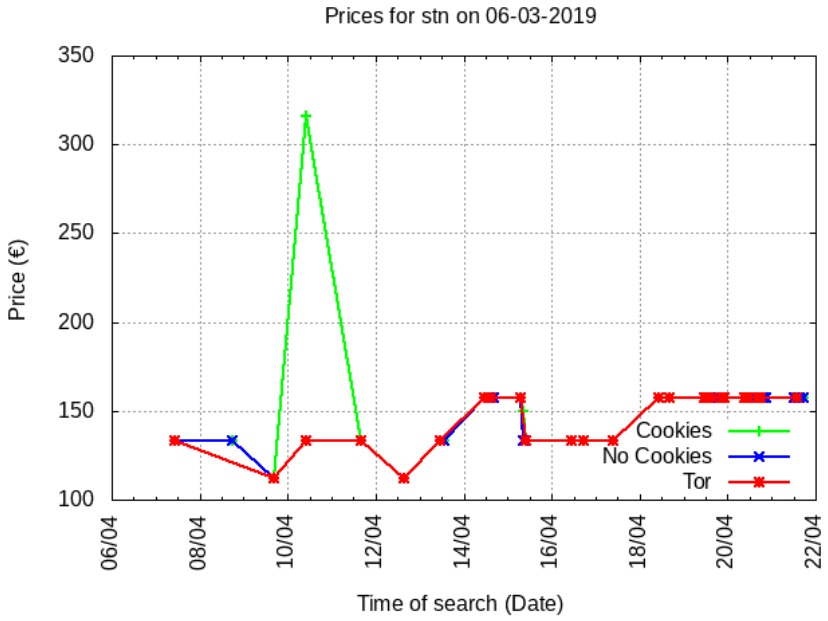
Solo dispongo de una posible hipótesis para estas subidas de precios. **Las subidas de precios para usuarios sin privacidad parecen estar relacionadas con el estado del país al que se viaja**.

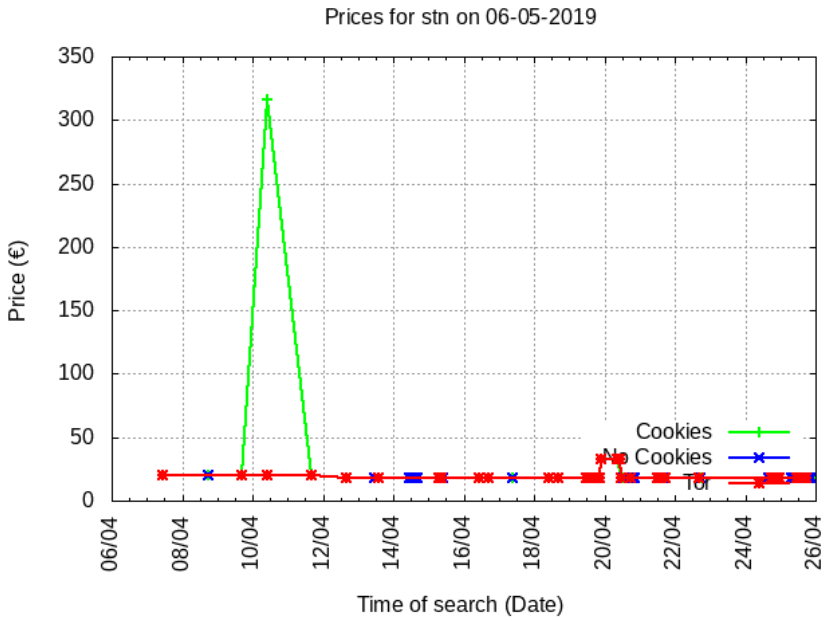
De los vuelos destacados de Stansted (Londres), ocurre lo siguiente: Todos estos incrementos puntuales tienen lugar el 10 de Abril. En búsqueda de si la fecha pudiera tener correlación con este hecho, nos encontramos con que el 10 de Abril el Parlamento Eu-

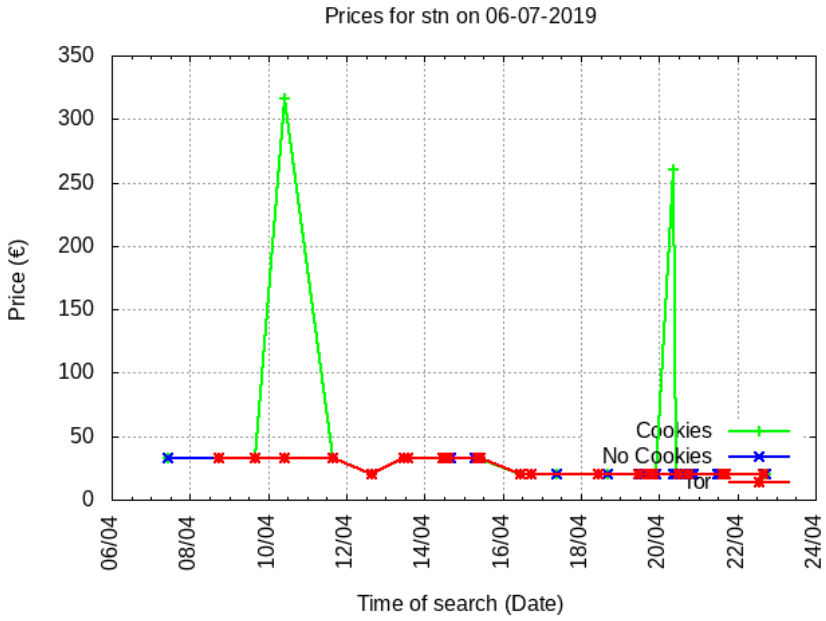
ropeo convocó una reunión sobre el Brexit, donde se dio un últimatum a Gran Bretaña (a la par que esta bajaba en bolsa).

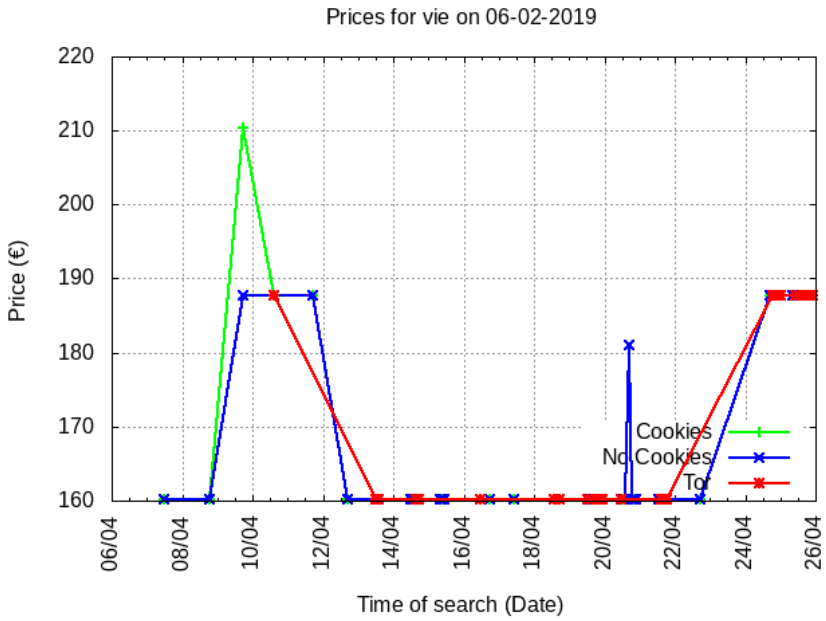


Lo más probable es que no haya correlación en estos hechos, pero con toda la opacidad que encontramos al intentar dar una respuesta, cualquier pequeña relación se debe dejar apuntada de cara a futuras investigaciones.


Prices for stn on 06-03-2019
Price (€)
350
300
250
200
150
100
06/04
08/04
10/04
12/04
14/04
16/04
18/04
20/04
22/04
Time of search (Date)
Cookies
No Cookies
Tor


Prices for stn on 06-05-2019
Price (€)
350
300
250
200
150
100
50
0
06/04
08/04
10/04
12/04
14/04
16/04
18/04
20/04
22/04
24/04
26/04
Time of search (Date)
Cookies
No Cookies
Tor


Prices for stn on 06-07-2019
Price (€)
350
300
250
200
150
100
50
0
06/04
08/04
10/04
12/04
14/04
16/04
18/04
20/04
22/04
24/04
Time of search (Date)
Cookies
No Cookies
Tor


Prices for vie on 06-02-2019
Price (€)
220
210
200
190
180
170
160
06/04
08/04
10/04
12/04
14/04
16/04
18/04
20/04
22/04
24/04
26/04
Time of search (Date)
Cookies
No Cookies
Tor

Síntesis: Respuesta y Posibles Vías

Con ello, llegamos a la respuesta:

> **¿Las aerolíneas principales del mercado aplican discriminación de precios de primer grado a sus usuarios de Internet?**

La respuesta (al parecer) es un **"No"** o, al menos, **no lo suficientemente significativa**:

- La media de precio en vuelos encontrados apenas difiere del euro entre los tres métodos.
- Apenas hemos encontrado discrepancias y, al haberlas, incluso teníamos casos en los que llegaban a salir perjudicados los métodos de mayor privacidad.
- Hemos encontrado muchos vuelos que el precio ha ido aunado a los tres métodos de búsqueda, tanto variando como quedando fijo el precio.

Son unas **10000 búsquedas** las que sustentan esta afirmación, pero la calidad tiene tanta importancia como la cantidad. Recordemos que:

- Echamos un vistazo a estudios previos y tampoco han conseguido dar luz (si bien con menor base empírica de la que tenemos ahora).
- Hemos leído sobre legalidades y solo encontrado incertidumbre en la discriminación de primer grado.
- Hemos analizado el tráfico usual entre un cliente y el host web. Aquí encontramos bastantes hosts secundarios de marketing y data analysis, con variables en sus cookies que podían inducir a la discriminación de primer grado (pero no teníamos muestras claras de su funcionamiento, era mera especulación).
- Con todo esto, hemos codificado nuestra base empírica y esta es la autora de esas 10000 búsquedas.

Por otro lado, como no todo ha de ser negro tras todo el trabajo realizado, dejo constancia de **algunos puntos que pueden dar bastante luz a la pregunta**:

- En `flight_search/selenium` está todo el código fuente. Quizás haya algún fallo semántico esperando a ser visto.
- En el capítulo "6. Previa a Conclusiones" hay bastantes consideraciones a tener en cuenta que podrían alterar los datos de la respuesta: ¿Incluir servidores de fuera de Europa en Tor?, ¿distinguir los vuelos buscados también por la hora en

lugar de centrarnos en el más barato? ¿Intentar (si bien es bastante complejo) tener constancia de los asientos restantes?

- ¿Y si usáramos `flight_search` en un ordenador más potente? Ejecutar durante todo el día `flight_search`, ampliando implementación de otras aerolíneas (que no nos pidan *captchas*), asegurándonos de *Tor Relays* que no estén bloqueados, buscar el mismo vuelo muchas más veces al día…

Y lo último y más importante:

- ¿Cuál es la causa de esas subidas puntuales de 300 euros?

Todo esto son ideas (quizás algunas lejanas a mi alcance) que conseguirían dar bastante mayor base empírica a esta pregunta que bien sabía que iba a tener (de momento) una difícil respuesta.

A

Bibliografia

1. Introducción

"¿Qué es la discriminación de precios y por qué es tan importante?", InformaBTL, Alejandro Ramírez: https://www.informabtl.com/la-discriminacion-precios-tan-importante/

"Do airlines use cookies to increase prices for flights?", The Sun, Hollie Borland: https://www.thesun.co.uk/money/5482811/do-airlines-hike-prices-if-you-keep-looking-at-the-same-flights-on-the-same-day/

"The Truth About Whether Airlines Jack Up Prices If You Keep Searching The Same Flight", TIME, Lucinda Shen: http://time.com/4899508/flight-search-history-price/

2. Legalidades

"Ley de Propiedad Intelectual", BOE, Ministerio de Cultura, 22 de Abril de 1996: https://www.boe.es/buscar/pdf/1996/BOE-A-1996-8930-consolidado.pdf

"La Ingeniería Inversa es Legal en España", Abanlex, Rafael Soria Ruiz, 9 de Diciembre de 2013 https://www.abanlex.com/2013/12/la-ingenieria-inversa-es-legal-en-espana/

"Política de protección de datos personales", Iberia: https://www.iberia.com/es/informacion-sobre-privacidad/

"Información Legal", Iberia: https://www.iberia.com/es/informacion-legal/

"Nuestra Política de Cookies", Ryanair: https://www.ryanair.com/es/es/empresa/cookies

"Derecho de supresión ("al olvido"), AEPD (Agencia Española de Protección de Datos): https://www.aepd.es/areas/internet/derecho-al-olvido.html

"¿Las cookies incrementan los precios de los vuelos? ¡Te lo contamos!", Skyscanner, Patricia Cuni, 5 de Abril de 2016: https://www.skyscanner.es/noticias/consejos/las-cookies-incrementan-los-precios-de-los-vuelos-te-lo-contamos

"Política de Cookies", Skyscanner: https://www.skyscanner.es/prensa/politica-de-cookies

"Política de Privacidad", Skyscanner, 15 de Mayo de 2018: https://www.skyscanner.es/prensa/politica-de-privacidad

3. Primer Contacto

"Getting Started with Burp Suite", Porswigger: https://portswigger.net/burp/documentation/desktop/getting-started

"IBM Tealeaf", IBM: https://www.ibm.com/us-en/marketplace/session-replay-and-interaction-analytics?loc=es-es

".krxd.net", CookiePedia: https://cookiepedia.co.uk/host/.krxd.net

"Cómo funciona la publicidad de internet, tercera parte: seguimiento de usuarios", KasperkyLab, Flavio Negrini: https://www.kaspersky.es/blog/internet-ads-103/9648/

Página web de Criteo: https://www.criteo.com/es/

Página web de Flocktory: https://www.flocktory.com/en/

"Oracle Maxymiser: Testing and Personalization" de Oracle: https://www.oracle.com/es/marketingcloud/products/testing-and-optimization/

Página web de LaunchDarkly: https://launchdarkly.com/

Página web de AdRoll: https://www.adroll.com/

Página web de AdobeDTM: https://dtm.adobe.com/

"IP Lead Tracking", ModernMarketing, 11 de Noviembre de 2017: https://www.modernmarketingpartners.com/ip-lead-tracking-just-the-facts/

"La guía del A/B testing", AB Tasty: https://www.abtasty.com/es/ab-testing/

4. Requests

Dado que el capítulo es orientado a la implementación, los recursos listados a continuación se tratan de "ayudas utilizadas" antes que "referencias":

"Practical Introduction to Web Scraping in Python", Real Python, Colin Okeefe: https://realpython.com/python-web-scraping-practical-introduction/

"Requests: HTTP for Humans", Requests 2.21.0 documentation: https://2.python-requests.org//en/master/

"Requests vs Selenium Python", StackOverflow, 14 de Octubre de 2017: https://stackoverflow.com/questions/46746085/requests-vs-selenium-python

"Beautiful Soup Documentation", Beautiful Soup 4.4.0 documentation: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

5. Selenium

Dado que el capítulo es orientado a la implementación, los recursos listados a continuación se tratan de "ayudas utilizadas" antes que "referencias":

"WebDriver API", Selenium Python Documentation: https://selenium-python.readthedocs.io/api.html

"Selenium Webdriver using Python: Tutorial", Guru99: https://www.guru99.com/selenium-python.html#5

La siguiente página web la considero de las más importantes para quien quiera profundizar en web scraping:

Página web de ScrapeHero: https://www.scrapehero.com/

Destaco los siguiente ejemplos y artículos:

- "How to prevent getting blacklisted while scraping", ScrapeHero: https://www.scrapehero.com/how-to-prevent-getting-blacklisted-while-scraping/
- "How to Scrape Flight Schedules and Prices from Expedia.com using Python and LXML", ScrapeHero: https://www.scrapehero.com/scrape-flight-schedules-and-prices-from-expedia/
- "How To Make Anonymous Requests using TorRequests and Python", ScrapeHero: https://www.scrapehero.com/make-anonymous-requests-using-tor-python/

"Selenium with Tor Browser using Python", A Medium Corporation, Manivannan Murugavel, 31 de Mayo de 2018: https://medium.com/@manivannan_data/selenium-with-tor-browser-using-python-7b3606b8c55c

"Tor FAQ", TorProject: https://2019.www.torproject.org/docs/faq.html.en

"Abuse FAQ", TorProject: https://2019.www.torproject.org/docs/faq-abuse.html.en

Interesante infografía sobre anonimidad y privacidad de Tor:

"How HTTPS and Tor Work Together to Protect Your Anonymity and Privacy", EFF: https://www.eff.org/pages/tor-and-https

6. Previa a Conclusiones

"Los precios de Skyscanner explicados en detalle", Skyscanner: https://www.skyscanner.es/noticias/caracteristicas/los-precios-de-skyscanner-explicados-en-detalle

"Cómo funciona Kayak", Kayak: https://www.kayak.es/company

"Robots.txt", Search Console, de Google: https://support.google.com/webmasters/answer/6062608?hl=es

7. Conclusiones

"La UE convoca una cumbre de urgencia el 10 de Abril", La Vanguardia, Jaume Masdeu: https://www.lavanguardia.com/internacional/20190329/461323562983/ue-cumbre-10-abril-brexit.html