# Asignación Eficiente de Votantes a Escuelas mediante Programación Matemática: El Caso de la Ciudad de Pergamino

Autor: Jeremías Lenzi
Director: Guillermo A. Durán
Director: Nicolás E. Stier-Moses
Co-Director: Damián Fernandez

31 de Mayo de 2013

**Resumen**

Este trabajo estudia el problema de asignar votantes a escuelas en una elección. Para ello, se consideran las distancias hasta cada escuela posible y los tiempos de espera en cada escuela según la cantidad de votantes asignados. Utilizando herramientas de optimización combinatoria, teoría de filas y teoría de grafos, se desarrolló un algoritmo de programación matemática para encontrar una solución eficiente. Se eligió la ciudad de Pergamino, provincia de Buenos Aires, Argentina, de la cual se extrajeron datos reales de distancias, densidades poblacionales, capacidades de mesas de votación, tiempos aproximados y distribución de votantes en cada cuadra, entre otros datos. La solución propuesta podría ser usada en los comicios para reducir el tiempo medio de espera de los votantes, evitando aglomeraciones en las horas pico y reduciendo la distancia que deben recorrer los votantes para llegar a las escuelas asignadas.

**Clasificación** (Math. Subject Classification)

90B30 - Operations research, mathematical programming.
90C90 - Operations research, mathematical programming. Applications of mathematical programming.
65K05 - Numerical Analysis. Mathematical programming methods.

**Palabras Claves**

-Investigación de Operaciones
-Optimización Lineal Entera
-Teoría de Filas

# Índice general

# Introducción

Este trabajo surge de la siguiente motivación, queríamos mostrar con un ejemplo, que existen procesos frecuentes que pueden ser drásticamente mejorados con la utilización de herramientas computacionales y de investigación operativa. Existen muchas actividades en nuestro país que todavía utilizan herramientas viejas y no aprovechan las ventajas que se podrían generar modernizándolas. Básicamente queremos mostrar un ejemplo de las ventajas de utilizar este tipo de herramientas en la vida real, cosa que no está bien aprovechada. Es así que en la actualidad en nuestro país hay muchos procesos que son generados manualmente. Un ejemplo, que es el que utilizamos nosotros, es el proceso de distribución de votantes en sus respectivos centros de votación en una elección.

En países democráticos suelen haber elecciones frecuentemente, tanto a nivel nacional, provincial y municipal. En las votaciones de Argentina, cada votante es asignado a un centro de votación—normalmente escuelas—en donde está obligado por ley a presentarse a votar en el día de la elección. Cada escuela tiene una cantidad fija de *mesas*, que son aulas de la escuela acondicionadas como cuarto oscuro. Cada mesa tiene la misma cantidad de votantes asignados, aunque la cantidad es elegida por el ente encargado de la realización y planificación de la elección.

De ahí surge la idea de generar un modelo de asignación de votantes a sus lugares de votación de manera eficiente. Con ésto me refiero a poder dar una distribución que minimice tiempos promedios de todo el proceso, además de dar la estructura para poder modificar los objetivos del modelo. El tiempo usado para el proceso de votación puede ser modelado por cuatro partes: el tiempo que tardaría la persona en ir desde su casa hasta el lugar de votación (la escuela), el tiempo que tardaría esta persona en esperar en la fila de la mesa de votación que fue asignada y entregar datos personales, el tiempo que tardaría en entrar al cuarto oscuro y emitir su voto, y finalmente el tiempo que tardaría en volver desde la escuela hasta su casa. La suma del tiempo destinado a cada etapa es el tiempo que tarda cada persona en votar.

Es importante proporcionar un buen modelo para este proceso pues existe una relación entre la decisión de una persona en votar y el tiempo que tardaría esta persona en los cuatro pasos mencionados anteriormente. Puesto que la legitimidad democrática del resultado de una elección está ligada fuertemente al porcentaje de

votantes efectivos, proporcionar un método que mejore este aspecto es muy importante. Otro aspecto que también aportaría a la legitimidad del proceso electoral es el hecho de proporcionar un método objetivo para la distribución, impidiendo así manipulaciones del tipo conocido como gerry-mandering, esto es la manipulación de las circunscripciones electorales de un territorio, uniéndolas, dividiéndolas o asociándolas, con el objeto de producir un efecto determinado sobre los resultados electorales, en el artículo [9] se puede encontrar información más detallada . También el hecho que manden a votar a lugares lejanos a personas de ciertas zonas que presentan un alto porcentaje de intención de voto a cierto partido político o candidato, disminuyendo así el porcentaje de personas que votan (teniendo en cuenta que existe relación entre distancia del lugar de votación y porcentaje de votantes efectivos). Porúltimo, está sucede que muchas personas no se quedan a votar si la fila es demasiado larga, ya sea porque tienen que trabajar (por mas que se declare día no laborable, muchas personas tienen trabajos los cuales le requieren que estén presentes ese día también) o por impaciencia.

El trabajo consta de 3 capítulos. El Capítulo 1 se basa en todo lo que se refiere al problema en sí. La explicación del caso particular de la Ciudad de Pergamino, Buenos Aires, Argentina, que es de donde tomamos los datos reales. Explicamos el método actual utilizado en esa ciudad. Detallamos el proceso de armado del modelo propuesto.

En el Capítulo 2 damos todo el marco teórico que utilizamos en el desarrollo del modelo. Explicamos Teoría de filas y los componentes que mas utilizamos. También explicamos un poco de optimización lineal-entera y brevemente como funciona el software utilizado para resolver el problema (Cplex).

Finalmente en el Capítulo 3 vemos todo el proceso de colección de datos. El filtrado del padrón electoral de la Ciudad de Pergamino y los distintos algoritmos que se hicieron para el armado de los parámetros del problema. Como se bajó toda la información del GoogleMaps utilizando el API de Google. Se detallan los resultados obtenidos luego de correr el modelo, y su comparación con los resultados utilizando el método actual de distribución. Además de todas las complicaciones que tuvimos durante el proceso.

# Capítulo 1

# Descripción del Problema y Modelo

## 1.1. El Problema

El problema básicamente consiste en que tenemos una elección en una ciudad, se tiene un conjunto de personas (votantes) y un conjunto de centros de votación (escuelas). Debemos entonces asignar a cada votante a una escuela para que pueda votar. Ahora, nosotros queremos generar una asignación que sea eficiente en el sentido que podamos minimizar el tiempo promedio que tardaría una persona en todo el proceso de votación. Éste consiste de las siguientes partes: el tiempo que tardaría una persona en ir desde su casa hasta el lugar de votación (la escuela), el tiempo que tardaría esta persona en esperar en la fila de la mesa de votación que fue asignada y entregar datos personales, el tiempo que tardaría en entrar al cuarto oscuro y emitir su voto, y finalmente el tiempo que tardaría en volver desde la escuela hasta su casa. La suma del tiempo destinado a cada etapa es el tiempo que tarda cada persona en votar. Nuestro objetivo es encontrar una asignación de votantes a escuelas que minimice el tiempo total promedio de votación por persona. Para esto proponemos plantear la posibilidad de tener una cantidad distinta de votantes asignados a cada mesa. Podría ser beneficioso en algunos casos subir un poco el tiempo de espera en la mesa para compensar que una escuela esté más cerca o viceversa, asignar a alguien a una escuela un poco más lejos si eso se compensa con que allí haya que esperar un poco menos.

## 1.2. Caso de Estudio: Pergamino, Provincia de Buenos Aires, Argentina

La ciudad de Pergamino es localizada en el centro de la Provincia de Buenos Aires y cuenta con aproximadamente 95.000 habitantes, de los cuales 55.000 están habilitados para votar. Se eligió Pergamino como primer caso de estudio por la disponibilidad de los datos y porque cuenta con una superficie y una densidad poblacional parecida a algunas zonas electorales de ciudades grandes.

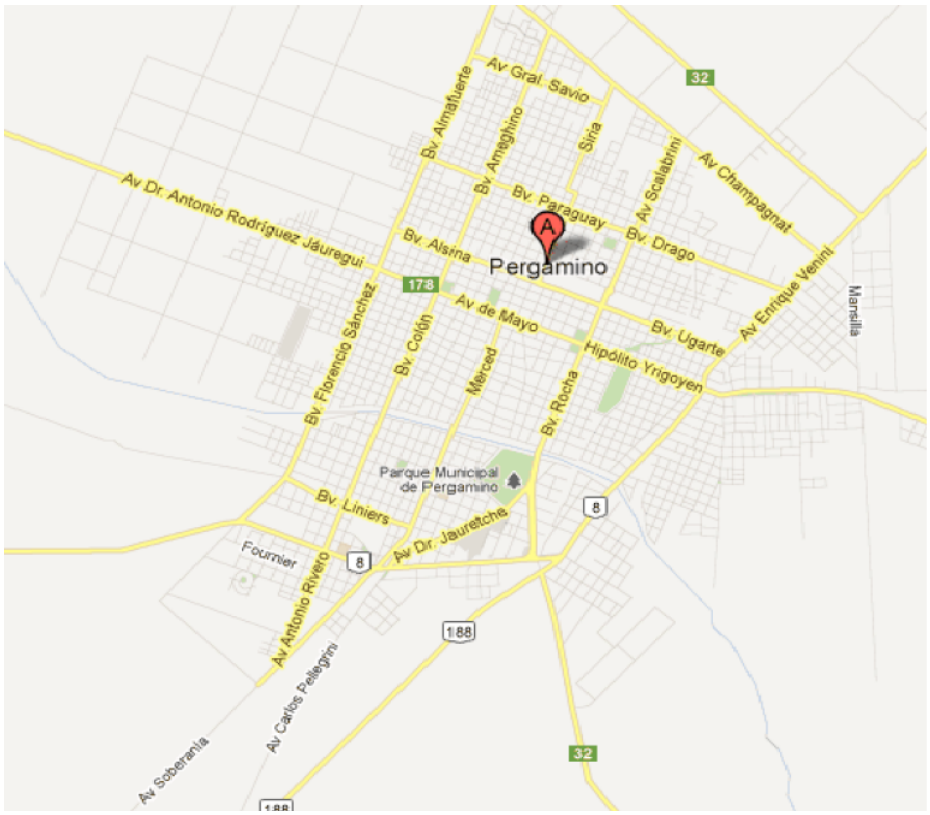


Figura 1.1: Pergamino, Provincia de Buenos Aires

En Pergamino hay 21 centros de votación ubicados en escuelas. Dichos centros están distribuídos geográficamente como lo indican los cuadrados negros en la Figura (1.2).

## 1.3. Distribución Actual de Votantes

En Argentina, el ente encargado de hacer la asignación de votantes a escuelas varía según la elección y el municipio. En municipios grandes, se asigna cada votante a una zona electoral, y luego los votantes de cada una de éstas son distribuídos en las escuelas dentro de esa zona. En municipios pequeños y medianos, no se realizan particiones y se asignan directamente los votantes a escuelas.

Una forma común de asignar votantes a escuelas es ordenar alfabéticamente al padrón de votantes por apellido y luego ir asignando grupos de votantes a escuelas hasta completar la capacidad. Según este proceso, es normal que algunos votantes sean asignados a escuelas a varios kilómetros de distancia, habiendo otras escuelas más cercanas.

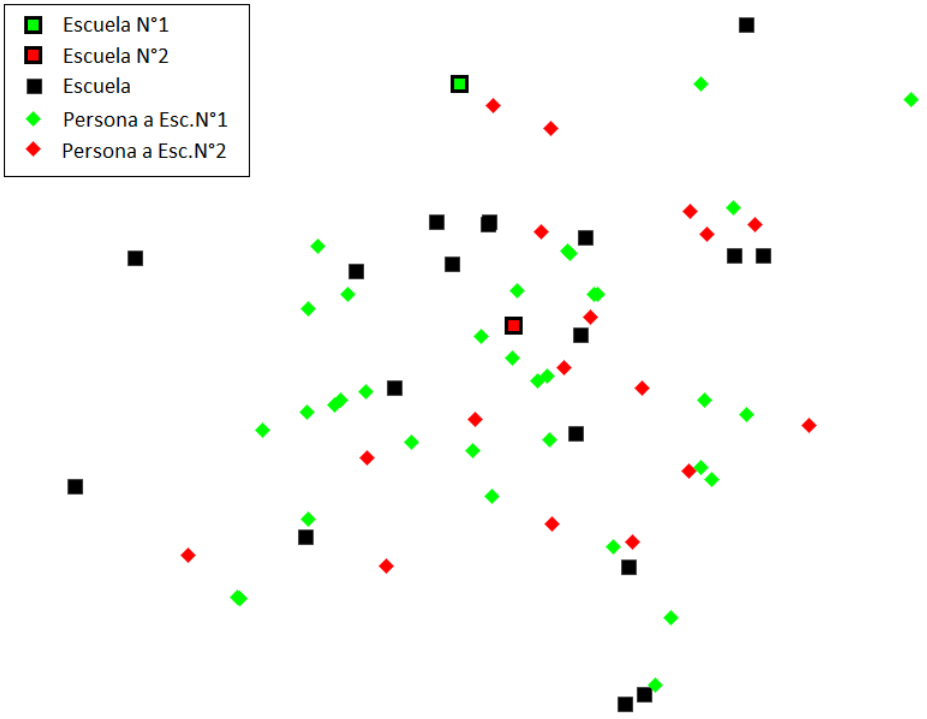


Figura 1.2: Ejemplo de la asignación real

La figura (1.2) muestra la dsitribución real de votantes a dos escuelas, que se muestran como cuadrados rojo y verde. En cuadrados negros se ubican las otras 19 escuelas de la ciudad. Los rombos rojos y verdes indican las direcciones de un conjunto de personas que fueron asignadas a votar en las escuelas de sus respectivos colores.

Se puede observar que aproximadamente la mitad de las escuelas están cerca del

centro del diagrama, donde existe una mayor densidad poblacional, mientras que el resto de las escuelas se encuentran en la periferia. La Figura (1.2) muestra que entre los votantes asignados a una misma escuela (designados con un rombo, verde o rojo dependiendo de la escuela que fueron asignados a votar), algunos habitan a menos de 100 metros de la escuela mientras que otros habitan a más de 7.000 metros.

Como se comentó anteriormente, el proceso actual asigna una cantidad fija de votantes a cada mesa. En nuestro modelo proponemos flexibilizar este requerimiento para poder variar la capacidad de votantes en distintas escuelas. Ésto permite que asignemos a más personas a escuelas en lugares de mayor densidad poblacional, ahorrando tiempo de viaje para esos votantes, a cambio de un leve aumento en el tiempo de espera en fila (mientras más elevemos la capacidad de una escuela, mayor será el tiempo de espera promedio en fila por votante, y viceversa).

El no tener establecido un proceso concreto y objetivo de asignación de votantes a escuelas permite potencialmente la posibilidad de realizar un fraude encubierto en una elección a través de manipulaciones demográficas. Ésto se podría conseguir asignando a los votantes que están alineados con el organizador de las elecciones a escuelas convenientes, mientras que los votantes que no lo apoyen podrían estar asignados a escuelas lejanas. Por más que en Argentina es obligatorio votar, algunos no lo hacen y posiblemente haya algún grado de correlación entre el hecho de efectivamente votar con la conveniencia de concurrir a votar y la distancia a la escuela asignada.

El modelo actual de distritos y zonas se presta a tergiversaciones con fines políticos [2]. Ésta es una práctica conocida y documentada en el mundo. De hecho, el término *gerry-mandering* (en inglés) es usado en ciencia política para referirse a 'alteraciones de los límites de un distrito electoral para que un partido obtenga mejores resultados' [**?**]. Tener reglas claras para separar en distritos y zonas dificultan poder realizar manipulaciones de este tipo.

## 1.4. Construcción del modelo

En un principio empezamos proponiendo como solución un modelo que solo tenga en cuenta las etapas de traslado, permitiendo solo minimizar el tiempo que le tomaría a una persona ir hacia la escuela y volver a su casa. Éste es un problema típico de asignación y la única complicación que tendría es todo el proceso de recolección de datos para el input.

Luego vimos la posibilidad de modificarlo para poder tener en cuenta en el modelo otros tiempos involucrados en el proceso de votación. Vimos que si variamos la cantidad de gente que enviamos a una fila de una mesa de votación cualquiera, también variará el tiempo promedio de espera por persona en esa fila. Entonces pensamos en fijarnos cuáles son los factores que influyen en el tiempo de espera en la fila. Por ende recurrimos a la teoría de filas, que será explayada en el Capítulo 3.

Vimos que podíamos modelar estas filas [6] tratándolas a cada una como como una sola fila, de un solo servicio, que vendría a ser la toma de datos, entrar al cuarto oscuro y votar. Además, el primer votante que llega es el primero en ser atendido y tanto las distribuciones de servicio como la de llegada de los votantes son Markovianas, ésto quiere decir que la probabilidad de que lleguen $n$ clientes en un intervalo de tiempo $t$ es:

$$\frac{(\lambda t)^n}{n!} e^{-\lambda t}$$

Decidimos que no íbamos a tener en cuenta para un primer modelo distinción de tráfico de gente. En ese momento se nos ocurrío que podíamos aplicar el modelo haciendo depender a la distribución de llegada del tiempo del da, osea que cada hora dentro del transcurso de la votación tenga una distribucin de llegada distinta, esto ser explicado con mas detalle en la sección de Trabajo a Futuro.

En el siguiente gráfico podemos ver una representación de una fila como las que utilizaremos. Donde $\mu$ representa el parámetro de la distribución de servicio y $\lambda$ el parámetro de la distribución de llegada, ésto está explicado detalladamente en el Capítulo 3.


Figura 1.3: Fila que modela una mesa de votación

Existe para este tipo de filas una fórmula que determina el tiempo promedio $W$ que tardaría una persona en esperar en la fila y pasar por el servicio. La formula es la siguiente.

$$W = \frac{1}{\mu - \lambda}, \qquad \rho = \frac{\lambda}{\mu} < 1 \tag{1.1}$$

El valor de $\rho$ nos indica en que parámetros de $\mu$ y $\lambda$ la fila no se sature, osea que no se empieza a acumular gente indefinidamente. Notemos que las únicas variables son la distribución de llegada y la distribución de servicio. éstas a su vez pueden ser representadas de la siguiente manera.

$$\mu = \frac{1}{\text{tiempo promedio en emitir voto}}, \qquad \lambda = \frac{\text{cantidad de personas asignadas a la mesa}}{\text{tiempo total}}$$

Por ende, la fórmula del tiempo promedio que tardaría una persona esperando en

una fila y votar depende de tres factores. La cantidad de tiempo total en que se puede ir a votar, la cantidad de personas asignadas a una fila en una mesa de votación y el tiempo promedio que se tarda una persona en votar dentro del cuarto oscuro. No tuvimos en cuenta el tiempo que tardarían los fisacales de mesas en tomar los datos de las personas, puesto que este proceso lo realizan mientras hay una persona dentro del cuarto oscuro.

De estos tres factores, solo podíamos tener influencia en uno de ellos, en la cantidad de personas que son designadas a votar en cada mesa de votación. Originalmente, osea en la asignación real, cada mesa tiene asignadas trecientas cincuenta personas. Decidimos poder variar este número para poder así asignar mas o menos gente en escuelas estratégicas, ya sean escuelas céntricas o escuelas periféricas, y controlar con la fórmula (1.1) el tiempo promedio por persona para que no asignemos personas de más a una escuela y éste se sature (notemos en el gráfico que hay una asíntota en 500 personas asignadas a la mesa, pero tengamos en cuenta que es muy complicado que vaya a votar el 100 % del padrón). Para simplificar las cosas, asignamos a todas las mesas de votación de una escuela la misma cantidad de personas, por ende esta cantidad va a depender de cada escuela.

De esta forma, jugamos con lo siguiente, si asignamos muchas personas a una escuela, esas personas van a tener que esperar más tiempo en la fila y si asignamos menos personas, éstas tendrán que esperar menos tiempo, todo dependiendo de la fórmula (1.1). Pero, se podría tener una ventaja en distancia ganada. Con ésto me refiero a que si puedo enviar a una persona a votar en una escuela que le quede más cerca que otras y el tiempo que gana es mayor al tiempo que tardaría esa persona en esperar en una fila de esa escuela, que al estar maś poblada va a ser mayor, entonces estaríamos ahorrando tiempo para esa persona. Podemos entonces ver que para escuelas en zonas de mayor densidad poblacional, nos conviene a priori asignar más personas y en escuelas de zonas con baja densidad poblacional nos convendría asignar menos personas.

Construímos un modelo de optimización lineal entera combinandoló con teoría de filas. Con el término Programación lineal entera, (PLE), nos referiremos a problemas que formalmente son problemas de programación lineal (pues se trata de una función lineal), Max ó Min $Z = Ax = b, \quad x \geq 0$, pero en los que algunas variables están restringidas a tomar valores enteros. Con teoría de filas nos referimos a la teoría que hay detrás de modelar los sistema de filas, los cuales se pueden describir como: clientes que llegan buscando un "servicio", esperan en éste, y abandonan el sistema una vez han sido atendidos. Éstos dos temas serán explicados con más detalle en el Capítulo 3. Ahora, tenemos el inconveniente que la fórmula (1.1) que indica el tiempo promedio de espera en fila por persona es no lineal. Para solucionar este inconveniente y para seguir con un modelo lineal por una cuestión de simplificación, aproximamos la función por una recta. Tomando como intersecciones de la recta y la función no lineal, a los puntos equivalentes a doscientas y cuatrocientas personas

como lo indica el gráfico (1.4). Notemos que la recta es cota superior en ese intervalo y por ende estaríamos tomando siempre valores superiores a los reales.

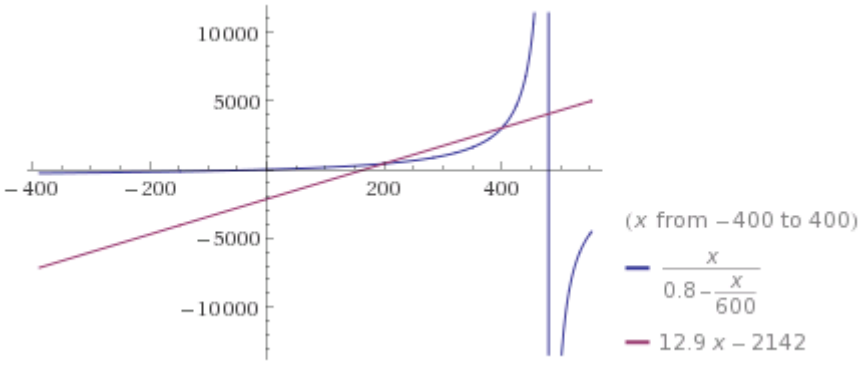


Figura 1.4: Interpolación de la función $W$

Utilizamos teoría de filas porque nos provee de las fórmulas que nos hace falta para modelar las mesas de votación. Y utilizamos optimización lineal entera puesto que optimización no lineal entera es demasiado compleja y por ende o se lleva lo entero a contínuo o lo no lineal a lineal. Es viable usar optimización lineal entera puesto que la aproximación de la función no lineal con la que cuenta el modelo es relativamente buena en realación a los valores que toman los tiempos promedios de votación.

Para el armado de restricciones que debería tener el modelo, propusimos las que eran obligatorias de acuerdo a la realidad. Osea, que todos los votantes sean asignados a alguna escuela, que cada votante se asignado a exactamente una escuela, que cada mesa de cada escuela se le asignen entre doscientas y cuatrocientas personas. Además, pensamos que era mejor para la futura aplicación del modelo, presentar una especie de equidad entre los votantes de una misma cuadra. Con esto nos referimos a que podamos generar alguna restricción talque permita que los votantes de una misma cuadra no vayan a votar muy lejos entre sí. Ésto permitiría que no haya disconformidades con el resultado de la distribución entre los vecinos. Ésto nos causó un poco de dificultad para implementarlo de una manera no tan complicada, puesto que si lo hacíamos comparando votante por votante, las restricciones iban a ser como en (1.2) e iban a ser un número combinatorio muy grande por lo que no iba a dar la memoria RAM del cpu para soportar tal programa.

$$\sum_{j \epsilon J} d_{ij} x_{ijk} - \sum_{g \epsilon J} d_{ig} x_{igk} \leq Equidad, \ \forall i, \forall k \tag{1.2}$$

Por ende lo que propusimos es dar una restricción que permita generar para cada

persona dentro de cada cuadra un conjunto de módulo una constante arbitraria de otras personas que vayan a votar a menor distancia que cierta otra constante también arbitraria.

Para el armado de los parámetros tuvimos que hacer un par de aproximaciones para poder disminuir la cantidad de datos usados significativamente, en el Capítulo 3 detallaremos bien esta cuestión. Todo el armado de inputs fue la parte del trabajo que mayores dificultades presentó. No tuvimos en cuenta la dirección real exacta, sino que juntamos a todas las personas de una misma cuadra (y la cuadra de en frente), y les asignamos el punto medio de esa cuadra como la ubicación en el mapa de todas esas personas. Armamos entonces 3 matrices principales. Una del orden de 4000x2 donde la primera columna indica el número de cuadra y la segunda su respectiva cantidad de votantes. Otra del orden de 4.000x21 donde cada entrada indica el tiempo que tardaría una persona en ir caminando desde la cuadra i-ésima hasta la escuela j-ésima. La tercera es de 21x2 donde la primera columna indica el número de escuela y la segunda la cantidad de mesas de votación que posee cada escuela. Utilizamos el tiempo que tardaría una persona en ir caminando, con su trayectoria respetando la topología de la ciudad, hasta cada escuela porque iba a ser mas sencillo para un primer modelo. La idea es trabajar el tema del transporte en un futuro.

## 1.5. Modelo

En esta sección detallaremos el modelo propuesto explicando cada una de sus partes.

**Input:**

$i \quad \epsilon \quad I$=Cuadras
$j \quad \epsilon \quad J$=Escuelas
$k \quad \epsilon \quad K_i$=Personas que viven en la cuadra $i$
$d_{ij} \quad \epsilon \quad D$=Matriz distancias $\quad \forall i, j$
$c_j$=Cant. mesas en $j \quad \forall j$
$CAPsup_j = 400 c_j \frac{60}{100} \quad \forall j$
$CAPinf_j = 200 c_j \frac{60}{100} \quad \forall j$
$\omega_1, \omega_2 = 1 \quad$ (Pesos)
$t = 600 \quad$ (Tiempo total de la votación)
$\mu = 0,8 \quad$ (Distribución servicio)
$Equidad = 350, \qquad comp = 5$

Los parámetros $I$,$J$ indican el conjunto de Cuadras y Escuelas respectivamente, donde se tiene un total de 3824 cuadras y 21 escuelas. El Parámetro $K_i$, dependiente de cada cuadra (la $i$-ésima), denota el conjunto de personas que viven en la $i$-ésima cuadra. La cuadra con mayor cantidad de personas tiene 296 votantes, pero slo 3

cuadras superan los 100 votantes por cuadra. El promedio de votantes por cuadra es de 9,4.

Los valores $d_{ij}$ son las entradas de la matriz $D$, las cuales indican el tiempo mínimo que una persona tardaría en ir caminando, a una velocidad de 6km/h, desde la cuadra $i$-ésima hasta la escuela $j$-ésima, respetando la topología de la ciudad, esto es que una persona solo puede hacer su trayecto por las calles. Hicimos uso de una aplicación que tiene el GoogleMaps que permite calcular los trayectos mínimos de un lugar a otro. Explayaremos en el Capítulo 2 la obtención de estos datos y el armado de esta matriz.

El parámetro $c_j$, dependiente de cada escuela (la $j$-ésima), denota la cantidad de mesas de votación (y por ende filas) que hay en la escuela $j$-ésima. La escuela que más filas posee, tiene 18. El promedio de mesas por escuela es de 9. Con los $c_j$ podemos determinar los rangos de capacidades de cada escuela. Por ejemplo, una escuela que posee 10 mesas de votación, puede tener capacidad para receptar entre 2.000 y 4.000 votantes.

$CAPsup_j$ indica la capacidad máxima que la escuela $j$-ésima puede contener, este número es $c_j$ (cantidad de mesas de votación) multiplicado por 400 (cantidad máxima permitida por mesa), pero contiene un leve ajuste, multiplicamos este último valor por 0,6 puesto que por cuestiones de falta de información o datos indecifrables tomamos el 85 % del padrón. Pero a su vez, el padrón que teníamos estaba incompleto puesto que nos proporcionaba la información de los votantes efectivos (70 %). Ésto nos permitirá comparar con los resultados de la asignación real, que actúa con todo el padrón con solo tomar el $85\,\%del70\,\%$ (ésto es el 60 %) del tiempo promedio por persona. Básicamente es achicar la capacidad de cada escuela proporcionalmente a lo que se achique el padrón. Notemos que como el parámetro de distribución de llegada $\lambda$ depende de la cantidad de personas que yo asigne a cada escuela entonces al tomar el padrón completo la solución variará.

Análogamente al parámetro anterior, $CAPinf_j$ indica la cantidad mínima de votantes que la escuela $j$-ésima puede tener asignados, tomando 200 como cantidad mínima de votantes asignados a una mesa. Hicimos el mismo ajuste de multiplicar por 0,85.

Elegimos que se asigne a cada mesa entre 200 y 400 personas porque permite una cantidad mayor a 350 votantes, pero no tan grande como para que sea muy larga la espera en fila en algunas mesas. En ese sentido es prudente porque no se acerca tanto a la asíntota de la función $W$ (que indica el tiempo promedio de espera en fila y votando), la cuál es alcanzada a las 500 personas asignadas a una mesa. Además para no dejar obsoletas algunas mesas decidimos dejar a cada una con por lo menos un mínimo de 200 personas asignadas, sin embargo se podría modificar este parámetro para ver si quedan mesas o incluso escuelas que son prescindibles (lo que ahorraría dinero pues se necesitarían menos fiscales y menos gastos de seguridad).

El tiempo en el cual una persona puede ir a votar está denotado por $t$ y tiene un

valor de 600 min, osea 10 horas.

La distribución de servicio fue calculada tomando datos de una elección nacional reciente, donde especificaba que el tiempo promedio que una persona tarda en entrar y salir del cuarto oscuro es aproximadamente de 1,25 min, lo que implica que $\mu = 0,8$.

El parámetro *Equidad* es un valor elejido arbitrariamente e indica cuanto es la diferencia de tiempo de votación máxima que podría haber para poder formar un conjunto de a lo sumo *comp* cantidad de personas en una misma cuadra. Notemos que no todo valor de *Equidad* nos dá una solución posible, tomemos el ejemplo en abstracto de una cuadra con dos personas y que todas las escuelas tengan capacidad para una sola persona, además le agreguemos que todas las escuelas distan entre sí en $10min$ yendo a $6km/h$, tendríamos que ambas personas serían asignadas a votar en escuelas distintas y los valores $Equidad < 5min$ serían inviables. Por ende tuvimos que testear con el modelo ya hecho para dar un valor accesible, éste fue de 350seg.

**Variables de Decisión:**

$$x_{ijk} = \begin{cases} 1, & \text{Pers. } k \text{ de Cuadra } i \text{ vota en Esc. } j \\ 0, & \text{c.c.} \end{cases}$$

Básicamente tenemos que decidir a que escuela asignamos a votar a cada persona. Por ende construimos nuestras variables de decisión que dependan de cada persona en cada cuadra y a cada escuela. Tomamos entonces variables binarias $x_{ijk}$, que toman el valor 1 si se asigna a votar a la persona $k$ de la cuadra $i$ a la escuela $j$, y toma el valor 0 en caso contrario (osea si ese votante no es asignado a votar a la escuela $j$).

La cantidad de variables de decisión es de alrrededor de 800.000.

**Función Objetivo:**

La función objetivo a minimizar es:

$$Z = \omega_1 \sum_{i \epsilon I} \sum_{j \epsilon J} \sum_{k \epsilon K_i} 2 d_{ij} x_{ijk} + \omega_2 \sum_{j \epsilon J} W_j \overbrace{\sum_{i \epsilon I} \sum_{k \epsilon K_i} x_{ijk}}^{Pers.\ Asignadas\ a\ j}$$

Teoría de Filas nos dice que el tiempo de espera individual es

$$W_j = 60 \left( \mu - \frac{\sum_{i \epsilon I} \sum_{k \epsilon K_i} x_{ijk}}{tc_j} \right)^{-1} \tag{1.3}$$

El tiempo de espera total y su aproximación por una función lineal es:

$$\frac{W_j}{60} \sum_{i \epsilon I} \sum_{k \epsilon K_i} x_{ijk} \cong (12,9 \sum_{i \epsilon I} \sum_{k \epsilon K_i} x_{ijk} - 2142)$$

La función objetivo que usamos en la asignación consta de dos partes. La primera representa la suma total de tiempo que tardan todos los votantes en ir y volver a

la escuela asignada, tomando como velocidad de caminar $6km/h$. La segunda parte de la función objetivo representa la suma total de tiempo de espera en la fila más el tiempo de votar, según la función mencionada anteriormente. Los pesos $\omega_1$ y $\omega_2$ nos permitirán darle más importancia a cualquiera de estas partes.

La primera parte funciona de forma tal que si asignamos a la persona $k$ de la cuadra $i$ a votar en la escuela $j$ ($x_{ijk} = 1$) entonces se suma el valor $2d_{ij}$ que representa el tiempo que tardaría esta persona en ir y volver a esa escuela. Si no la asignamos a esa escuela ($x_{ijk} = 0$) entonces no se suma nada. Al hacer las sumatorias de todas las personas, de todas las cuadras y todas las escuelas, tenemos que vamos a poder contemplar todas las soluciones posibles.

La segunda parte funciona de forma similar a la primera, si la persona $k$ de la cuadra $i$ a votar en la escuela $j$ ($x_{ijk} = 1$), entonces se suma el valor $W_j$ (a continuación explico este valor) y si no es asignada a votar en esa escuela no se suma nada.

El valor $W_j$ representa el valor aproximado del tiempo promedio de espera en fila por persona en la escuela $j$. Recordemos que según teoría de filas este valor dependía de la cantidad de personas que nosotros asignemos a cada escuela y por ende varía según $j$. Podemos ver en (1.3) como hicimos para aproximar $W_j$ por una función lineal cuyas constantes fueron calculadas tomando los puntos de intersección entre la recta y la función no lineal en los puntos representativos a 200 y 400 personas.

Los parámetros $\omega_1$ y $\omega_2$ son simplemente pesos que le pusimos a cada parte de la función objetivo para poder ponderarlas de manera distinta y así darle más importancia a una o a otra en caso de ser necesario.

Tuvimos en cuenta de pasar todas las sumas en segundos, por eso tanto las entradas de la matriz $D$ como los valores de $W_j$ (multiplicamos por 60 para pasar de minutos a segundos) representan valores en segundos.

**Restricciones:**

Sujeto a:

$$\sum_{j \epsilon J} x_{ijk} = 1 \quad \forall i, k \tag{1.4}$$

(Que se asigne una escuela por persona).

$$\sum_{j \epsilon J} \sum_{k \epsilon K_i} x_{ijk} = |K_i| \quad \forall i \tag{1.5}$$

(Todos votan).

$$200 c_j \frac{60}{100} = CAPinf_j \leq \sum_{i \epsilon I} \sum_{k \epsilon K_i} x_{ijk} \leq CAPsup_j = 400 c_j \frac{60}{100} \quad \forall j \tag{1.6}$$

(Que se respete la capacidad por escuela).

Nos gustaría respetar que los votantes dentro de una misma cuadra vayan a votar relativamente cerca:

$$\sum_{j \epsilon J} d_{ij} x_{ijk} - \sum_{g \epsilon J} d_{ig} x_{ig(k+r)} \leq Equidad, \ \forall i, \forall k \leq (|K_i| - r), \ 1 \leq r \leq comp \quad (1.7)$$

$$\sum_{j \epsilon J} d_{ij} x_{ijk} - \sum_{g \epsilon J} d_{ig} x_{ig(k-|K_i|+r)} \leq Equidad, \ \forall i, (|K_i| - r) \leq k \leq |K_i|, \ 1 \leq r \leq comp \quad (1.8)$$

donde $Equidad = 350seg, \quad comp = 5$

Trabajamos en total con cinco tipos de restricciones (1.4-1.8). La primera restricción (1.4) asegura que asignemos a votar a cada votante a solo una escuela. Ésto lo logramos fijando cada variable $x_{ijk}$ en las $i$ y $k$ (que indica que se trata de un votante fijo), para luego sumar sobre todas las escuelas. Recordemos que cada variable sumaba a la función objetivo un 1 o 0 dependiendo si asignabamos a votar al votante a cierta escuela o no. Por ende si no pusiésemos esta restricción, daríamos la posibilidad que para un mismo votante haya más de un 1 entre las posibles escuelas y estaríamos así designando a votar a una persona a más de una escuela.

La segunda restricción (1.5), indica que todos los votantes votan y así nos aseguramos que ninguna persona quede sin ser asignada a una escuela. Notemos que la suma total sobre las variables fijando las $i$ ($i$-ésima cuadra) la condicionamos para que dé el número total de personas por cuadra.

La tercera restricción (1.6) controla la cantidad de personas que podemos asignar a cada escuela. Habíamos visto que cada hay escuelas con distinta cantidad de mesas de votación, por ende las escuelas poseen capacidades distintas y pueden recibir cantidades distintas de personas para que voten en esa institución. Matemáticamente lo que hace la fórmula del centro de la desigualdad es dado un $j$ fijo, sumar sobre $i$ y $k$ todas las variables, lo que nos dá la cantidad de personas que serían asignadas a votar en la escuela $j$. El lado derecho garantiza la cota necesaria para respetar las 400 personas por mesa que habíamos modificado al armar el modelo. La capacidad máxima $CAPsup_j$ de la $j$-ésima escuela está dada por el número de mesas de esa escuela $c_j$ multiplicado por 400 que es valor máximo de personas que puede haber por mesa. El lado izquierdo hace análogamente lo mismo proporcionando una cota inferior, donde 200 es el número mínimo de personas que pueden ser asignados a cada mesa.

La cuarta y la quinta restricciónes (1.7,1.8) son un poco más complicadas. Lo que queríamos lograr con el objetivo de poder brindar una mejor respuesta al momento de aplicar el modelo en la vida real, es que haya una especie de equidad entre los votantes de una misma cuadra y éstos no tengan que recorrer distancias muy dispares entre si. Por ende construimos esta restricción que lo que hace a grandes

rasgos es permitir en la solución, en cada cuadra y para cada persona la existencia de un grupo de votantes de esa cuadra tales que tengan que recorrer una distancia cercana a esa persona. El grupo es de tamaño *comp* y la cota es el parámetro *Equidad*. Matemáticamente se fija primero una cuadra $i$ y una persona $k$ dentro de esa cuadra, después se multiplica a la variable $x_{ijk}$ por $d_{ij}$ (el tiempo que se tardaría en ir desde esa cuadra hasta la escuela $j$), luego se suma sobre todas las escuelas. A ésto se le resta lo mismo pero cambiando la persona $k$ por la persona $k + r$ donde $1 \leq r \leq comp$. Ésto permite asignar a las personas $k+1, ..., k+r$ sin que éstas viajen por más tiempo que la constante *Equidad* comparado con $k$. Al repetir este proceso por cada persona en la cuadra obtenemos la configuración deseada. Notemos que si $(|K_i| - r) \leq k \leq |K_i|$ entonces tomamos una configuración que compare con las primeras personas de la cuadra (de manera cíclica) y por ende tenemos la segunda parte de esta restricción.

Es importante decir que el número de restricciones que tiene el modelo con los parámetros propuestos es de aproximadamente 235.000 y que incrementa o decrece drásticamente dependiendo del parámetro *comp*.

**Output:**

Las salida del modelo son variables de decisión que indican donde votan las personas del padrón que viven en cada cuadra de Pergamino. Entonces vamos a poder ver nuestra solución en un conjunto de 3,824 matrices de tamaño 21x3824x$|K_i|$ para todo $i \epsilon I$ con entradas unos o ceros, con exactamente un 1 en cada una de sus 21 columnas y con ceros en las demás. Recordemos que $K_i$ es el conjunto de personas en la cuadra $i$ y por ende $|K_i|$ nos indica la cantidad de personas que viven en esa cuadra.

## 1.6. Trabajo a Futuro

Los siguientes pasos consisten en un análisis más exhaustivo de la asignación de votantes en la ciudad de Pergamino y a continuación en la aplicación del modelo a la Ciudad Autónoma de Buenos Aires para poder tomar zonas con distintas densidades poblacionales. Además, planeamos modificar el modelo de las siguientes formas:
(I) Variar la cantidad de mesas por escuela, para obtener otra variable.
(II) Agregar la restricción que haga que nadie empeore su situación comparado con la escuela asignada actualmente, para que pueda tener buenas repercuciones el modelo al ser implementado.
(III) Agregar modos de transporte (las personas pueden viajar en auto o a pie), para dar una visión más realista del problema.
(IV) Considerar otras esperas como estacionamiento si el votante va en auto a la escuela, también para dar una modelo más realista.
(V) Considerar tasas de llegada a votar dependientes en el tiempo, para poder tener en cuenta que hay horarios pico que generan los mayores tiempos de espera en fila.

(VI) Analizar si sería más correcto en cuestiones de modelaje tomar una distribución de otro tipo en vez de una de Markov.

## 1.7. Resultados de la Asignación de Votantes Usando Programación Entera

Considerando una capacidad máxima de 350 personas por mesa y que cada persona tarda en promedio 1,25 minutos en el cuarto oscuro, la asignación real del padrón de Pergamino a escuelas obtiene un tiempo total de votación de 109.180.890 segundos, con lo que el 85 % de este valor es de 92.803.757 segundos. Recordemos que nosotros trabajamos con el 85 % del total del padrón.

A continuación presentamos algunos de los resultados que obtuvimos al resolver el programa con estos datos reales. Primero, variando la capacidad máxima de personas por mesa a 400, y manteniendo la duración promedio que tarda una persona en el cuarto oscuro, obtuvimos un tiempo total global de 44.635.955 segundos (una mejora del 52 %). Segundo, modelamos el sistema de boleta única puesta en marcha en recientes elecciones en la ciudad de Córdoba, Argentina que consiste en simplificar el sistema de boletas permitiendo además la utilización de varias mesas por aula en las escuelas (una fila por aula). Ésto nos permite entonces variar la capacidad por mesa a 600 y la distribución de servicio a 3,2 pues tendríamos 4 cuartos oscuros por fila. Obtuvimos un tiempo total global de 29.977.761 segundos (una mejora del 68 %). Cabe destacar que con la asignación real actual y cambiando al sistema de boleta única, sólo tendríamos una mejora leve en lo que que se refiere al tiempo que tardaría una persona en efectuar todo el proceso de votación

En definitiva, logramos al resolver el problema, reducir el tiempo promedio de votación por persona a la mitad, manteniendo el sistema actual de votación.

# Capítulo 2

# Construcción de Algoritmos para la Obtención de Datos

## 2.1. Recopilación de Información

Cuando teníamos que tomar la decisión de elegir una ciudad para poder utilizar los datos reales y armar el input de nuestro modelo, nos surgieron un par de dificultades burocráticas para poder obtener los padrones de alguna ciudad. Empezamos pidiendo los padrones de la Ciudad Autónoma de Buenos Aires y de Córdoba. Nos comunicamos con las autoridades encargadas de poder proporcionar tal información respectivas a cada ciudad. En el caso de Córdoba nos negaron el acceso al padrón y en la Ciudad Autónoma de Buenos Aires nos dijeron que iba a tardar el trámite, pero que íbamos a poder tener acceso al padrón de algunas zonas. Por una cuestión de tiempo se decidió empezar con los datos de una ciudad más chica que tenga una densidad poblacional semejante a la de las zonas de las ciudades grandes. Surgió la posibilidad de conseguir gran parte del padrón de la Ciudad de Pergamino, Provincia de Buenos Aires, la cuál se adecuaba con lo que buscábamos. Por ende nos pusimos a trabajar con el padrón de esta ciudad.

Los datos que íbamos a necesitar nosotros eran de acuerdo a dos cuestiones. La primera, para el armado del input de nuestro modelo ́bamos a necesitar cuáles iban a ser los votantes habilitados, las direcciones de cada uno de los votantes y las escuelas que iban a ser utilizadas como centros de votación. La segunda, para poder dar una noción de cuan efectivo es el modelo, necesitábamos comparar el resultado obtenido con la asignación real y por ende ́bamos a requerir de la última distribución real que se había efectuado. Ésto es, además de la información pedida en el párrafo anterior, la escuela donde fue asignada a votar cada persona en esa última elección. Finalmente pudimos obtener toda la información requerida para esta ciudad.

En esta sección vamos a explicar como hicimos para armar las tres matrices principales para el input de nuestro modelo. La matriz $D$ de tiempos en ir caminando

desde cada cuadra hasta cada escuela. La matriz que indica cuantas personas viven en cada cuadra. La matriz que indica cuantas mesas de votación hay en cada escuela.

Para lograr el armado de estas tres matrices tuvimos que filtrar los datos del padrón para que éstos puedan ser leídos por los programas. Explicaremos los pasos que tuvimos en cuenta para poder llegar al armado de las tres matrices mencionadas anteriormente.

El padrón original estaba en un archivo con formato binario de Excel, por lo que utilizamos hojas de cálculo para poder trabajar más facilmente con los datos. La información que proporcionaba el padrón eran "DNI","DIRECCIÓN","SEXO", "AÑO DE NACIMIENTO","ESCUELA ASIGNADA","DIRECCIÓN ESCUELA","MESA", "NÚMERO DE PERSONA EN LA MESA". El orden del padrón estaba dado por cada escuela, luego por el número de mesa y después por el número de persona en la mesa.

Lo primero que hicimos fue separar todas las listas en archivos distintos para asi poder manipularlas más facilmente. Una vez hecho esto la idea fue ir armando nuevas listas de acuerdo a la información que querámos extraer.

Luego enumeramos a cada escuela de acuerdo al orden en que venían en el archivo original. Hicimos lo mismo con las personas.

Para el armado de la matriz de cantidad de mesas por escuela utilizamos la listas de "MESA". Luego, creamos en Octave una función que nos dé el número que está antes de un uno y no sea un uno. Ésta función dá como output una lista de 21 elementos, donde cada uno de esos elementos indica el número de la mesa con mayor número dentro de las mesas de cada escuela puesto que estaban en orden ascendente por escuela, en otras palabras la cantidad de mesas por escuela.

Para el armado de la matriz que indica cuantas personas viven en cada cuadra tuvimos que efectuar varios pasos.

(M.I) Agarrar la lista de direcciones de las personas (respetando el orden anterior) y ubicarla en un archivo de hoja de cálculo.

(M.II) Utilizar la función "Sort" para poder ordenar todas las cuadras. Nos quedáron sublistas de las direcciones que compartían la misma calle, una debajo de la otra.

(M.III) Creamos una función que separe los números de las casas de las direcciones (éstos estaban ubicados luego del nombre de la calle). Entonces nos quedáron dos columnas, en una el nombre de la calle y en la otra el número de las casas.

(M.IV) Construímos otra función donde toma la segunda lista (la de los números) y, sin modificar su orden, reemplaza cada número $N$ por $\bar{N} = 100 \left\lfloor \frac{N}{100} \right\rfloor + 50$ donde $\lfloor \cdot \rfloor$ denota la parte entera. Por ejemplo, si $N = 987$ entonces obtendremos $\bar{N} = 950$. Vamos así a obtener una lista de números que terminan todos en 50. Notemos que todas las direcciones que están dentro de una misma cuadra van a tener la misma calle (entrada en la primera columna) y el mismo número terminado en 50 (entrada en la segunda lista).

(M.V) Unimos estas dos columnas en una sola.

(M.VI) Creamos otra función que permite por un lado contar la cantidad de entradas distintas que tiene la columna (ésto nos daría la cantidad de cuadras distintas que hay) y por otro lado que cuente cuantas veces se repite cada una de esas entradas (esto nos iba a proporcionar la cantidad de votantes que viven en cada cuadra). El output de esta función nos dá la matriz que necesitamos.

Para el armado de la matriz $D$ que indica el tiempo que tardaría una persona en ir a hasta cada escuela hicimos los siguientes pasos.

(D.I) Tomamos la lista "DIRECCIONES DE LAS ESCUELAS" y la ubicamos en un archivo de hoja de cálculo. Sin cambiarle el orden.

(D.II) Creamos una función que elimine toda entrada si ya existe otra anterior que coincida con ésta. Con esto nos generamos una lista sin repeticiones de las direcciones de las escuelas, que luego enumeramos.

(D.III) Tomamos la lista que habíamos efectuado en el paso (M.V) del armado de la matriz anterior y generamos una función que elimine todas las entradas de la columna que se repitan (dejando una de cada tipo). De este modo obtenemos la lista de las direcciones de los puntos medios de todas las cuadras. Luego enumeramos cada cuadra.

(D.IV) Aplicamos los algoritmos que explicaremos en la siguiente sección, donde utilizamos el API del GoogleMaps para poder combinar cada entrada de las dos listas que hicimos en los pasos (D.II) y (D.III) (direcciones de las escuelas y las cuadras).

(D.V) Toda la información obtenida del paso anterior la ubicamos en una matriz donde cada fila representa una cuadra y cada columna una escuela.

Finalmente obtuvimos las tres matrices principales que nos hacían falta para poder hacer correr el modelo. Tuvimos varias dificultades en este proceso, pero las explicaremos en la última sección de este capítulo.

## 2.2. Algoritmos Usando el API de Google

Para poder armar la matriz que nos permitía ver cuanto es el tiempo que le llevaría a una persona ir desde el punto medio de cada cuadra hasta cada escuela yendo por el camino más corto respetando la topología de la ciudad, osea que se vaya caminando por las rutas marcadas por calles, lo que hicimos fue utilizar el API del GoogleMaps y generar un par de algoritmos que nos permitiesen extraer información del mapa, para más información revisar [5].

El GoogleMaps tiene una aplicación que te permite calcular las rutas más cortas desde un punto arbitrario del mapa a otro y también calcula el tiempo de recorrido ya sea a pie (6km/h) o en auto (en este último tiene en cuenta las direcciones de las calles como para no ir en contramano). Nosotros decimimos trabajar con recorridos a pie en una primera instancia y como trabajo a futuro está modificar este aspecto.

Haciendo uso de esta aplicación, creamos un algoritmo en bash que nos provee de una manera automática la asignación de los puntos de partida y llegada que le hace falta a la aplicación. Este algoritmo es el siguinete:

```
LISTA="01 02 03 04 05 06 07 08 09 "
LISTB=‘echo {10..21}‘
LISTC=‘echo {10.. 3824}‘
LISTD=$LISTA$LISTB
LISTE=$LISTA$LISTC

rm -f links.txt
for i in $LISTE
do
  for j in $LISTD
  do
    origin=‘head -$i DireccionCuadras.txt | tail -1‘
    destination=‘head -$j DireccionEscuelas.txt | tail -1‘
    echo "wget ’http://maps.googleapis.com/maps/api/distancematrix/json?
origins=$origin&destinations=$destination&sensor=false&mode=walking’
-O $i-$j.sh" chmod u+x $i-$j.sh >> links.txt
  done
done
chmod u+x links.txt
```

Donde *DireccionCuadras.txt* es la lista generada en el paso (D.III) del armado de la matriz $D$, ésta indica las direcciones de los puntos medios de cada cuadra donde existe un votante. Y *DireccionEscuelas.txt* es la lista del paso (D.II) del armado de la matriz $D$, ésta indica las direcciones de cada escuela.

Básicamente lo que hace el algoritmo es tomar una dirección de la lista de cuadras, una dirección de la lista de escuelas y nos devuelve un archivo ".json" en donde va a estar la información que queremos. Para extraer esa datos, lo que hicimos fue generar otro algoritmo en bash que nos permita sacar solo el tiempo (dado en segundos) de recorrido mínimo entre una cuadra y una escuela, para luego ubicar ese número en la entrada correspondiente de la matriz $D$. El algoritmo es el siguiente:

```
LISTA="01 02 03 04 05 06 07 08 09 "
LISTB=‘echo {10..21}‘
LISTC=‘echo {10.. 3824}‘
LISTD=$LISTA$LISTB
LISTE=$LISTA$LISTC

rm -f metros
```

```
rm -f segundos
for i in $LISTE
do
  for j in $LISTD
  do
      cat $i-$j.json|grep value|awk '{print $3}'|head -n1 >> metros
      cat $i-$j.json|grep value|awk '{print $3}'|tail -n1 >> segundos
  done
done
```

Este algoritmo también registra la distancia mínima entre cada cuadra y cada escuela, pero estos datos no serán utilizados. El output son dos listas, "metros" (indica distancias) y "segundos" (indica tiempos). Acá surge un inconveniente muy grande que es que el API del GoogleMaps no permite descargar más de 2.500 archivos ".json" por IP por día, siendo que nosotros necesitábamaos 80.304 archivos (la multiplicación entre la cantidad de cuadras y la cantidad de escuelas). Por ende tuvimos que ingeniárnosla para poder esquivar este inconveniente. Ésto será detallado en la sección "Dificultades y Observaciones: Filtrado del Padrón y Restricciones de Google".

El tiempo aproximado de descarga de 2.400 archivos ".json" (un número arbitrario menor a 2.500) es de una hora con diez minutos, todo esto desde el CPU y la conección de internet que teníamos nosotros. Por ende tuvimos que generar listas de 2.400 para poder aprovechar el tiempo mientras se descargaban los archivos. Tuvimos 34 tandas de archivos y por ende 34 listas "segundos". Una vez que tuvimos todas las unimos (respetando el orden) en un archivo A.txt y aplicamos el siguiente algoritmo en Octave:

```
load A.txt
B=reshape(A,3824,21);
save mat_segundos.txt
```

Donde A es la lista que es la unión de las 34 sublistas. Ésto lo que hace es armar la matriz $D$, debido a cómo habíamos ordenado los archivos ".json" al momento de crearlos.

## 2.3. Algoritmos para la Comparación de la Asignación Real con la del Modelo

Para poder ver la efectividad de la solución de asignación que nos dá el modelo, debemos compararla con la distribución real que está dada en el padrón. Para eso calculamos la suma total de tiempos de todas y todos los votantes de acuerdo con

la asignación que les correspondía en la última votación y le sumamos el tiempo promedio que tardarían en esperar en fila y votar (dado por la función $W_j$ tomando la variable $\lambda = \frac{350}{600}$, recordemos que en la votación real se le asignaba a cada mesa 350 personas). Luego la comparamos con el mínimo de la función objetivo dado al encontrar una solución al modelo (utilizando el Cplex encontramos esta solución, para más información sobre este prgrama ir a [3]).

Ahora, tuvimos que hacer una serie de pasos para obtener la suma de tiempos de todos los votantes (al efectuar todo el proceso de votación), los cuales fueron los siguientes:

(T.I) Juntamos en un archivo de hoja de cálculo a las listas de "DIRECIÓN" Y "ESCUELA ASIGNADA". Además tuvimos en cuenta la lista que indica cantidad de votantes que viven cada cuadra (que ya la habíamos construído) y la Matriz $D$ que indica el tiempo que tardaría una persona en ir desde cada cuadra hasta cada escuela yendo por el camino más corto posible. Estas últimas listas las utilizaremos como parte del input para el algoritmo del paso (T.III).

(T.II) Construimos una función que reemplace a cada dirección de votante por su número de votante (recordemos que tanto a los votantes como a las escuelas los habíamos numerado). Entonces nos va a quedar en el archivo, dos columnas cuyas entradas son números. En la primera números del 1 al 35.803 (número de votante) y en la segunda números del 1 al 21 dependiendo de la escuela a la cuál representen.

(T.III) Hicimos una función que permita sumar los valor $d_{ij}$ (entrada de la matriz $D$) sobre todos los $i$, tomando éstos como los valores de la primera columna de la lista generada en el paso (T.II) y también sobre los $j$, tomando éstos como las entradas $e_i$ de la segunda columna de la lista, correspondiente a cada $i$. Con ésto vamos a obtener la suma sobre todos los votantes de los tiempos que tardarían en ir por el camino más corto a la escuela que fueron asignados a votar en la elección real:

$$\sum_{i=1}^{35803} d_{i,e_i}$$

Además, le sumamos el tiempo que tardarían en esperar en fila y votar utilizando teoría de filas, como ya mencionamos. El algoritmo que efectúa el paso (T.III) es el siguiente:

```
total=0;
k=0;
c=0;
max=0;
num=0;
lambdaReal=350/600;
mu=1/1.25;
Wreal=60*1/(mu-lambdaReal);
```

```
Wtotal=0;

for i=1:size(CUADRAS(:,1))
    for j=1: CUADRAS(i,2)-1
        k=k+1;
        c=ESCUELA(k,1);
        total=total+2*MATRIZ(j,c)+Wreal;
        if(MATRIZ(j,c)>max)
                max=2*MATRIZ(j,c);
        endif;
        Wtotal=Wtotal+Wreal;
    endfor
endfor
max=max+Wreal;
save votacionReal.txt total max Wtotal MATRIZ
```

Donde "CUADRAS" es la matriz de dimensión 3.824x2 que tiene en la primera columna el número de cuadra y en la segunda columna tiene la cantidad de votantes que viven en esa cuadra, recordemos que esta lista ya la teníamos. La matriz "MATRIZ" es la matriz $D$. La matriz "ESCUELA" es simplemente la segunda columna de la lista del paso (T.II).

La función nos proporciona la suma de tiempo invertido "total" entre todos los votantes en la elección real. Además, nos dá el tiempo de recorrido "max" que tendría que hacer la persona que mas tarda de acuerdo a donde fue asignada a votar. También nos proporciona el tiempo "Wtotal" que es la suma de todas las esperas en filas.

## 2.4. Dificultades y Observaciones: Filtrado del Padrón y Restricciones de Google

En el transcurso del armado del modelo y de su implementación nos han surgido varias dificultades, sobre todo en el momento del armado del input del modelo pues teníamos que tomar datos tomados por varias personas que no tenían en cuenta una posible futura utilización de éstos en alguna herramienta computacional. Por ende nuestra mayor dificultad fue a la hora de traducir la información para que sea legible tanto por el API del GoogleMaps como por el modelo que propusimos.

Como ya comenté la principal dificultad que nos surgió en el transcurso de este trabajo fue el hecho que el padrón, osea los datos originales que nos entregaron las autoridades de la ciudad de Pergamino, no estaba preparado para ser utilizado como base de datos informático sino como información que iba a ser utilizada de forma manual en el momento de la repartición de lugares de votación. Por ende

la información estaba descuidada, había nombres de calles que estaban escritas de más de quince formas distintas, con abreviaciones distintas, algunas indicando que era avenida y otras no, etc. Básicamente esto hacía extremadamente dificil hacer un programa que registre todas estas formas como una misma calle.

Incluso el GoogleMaps no reconocía todas estas formas por lo que eso fue un problema doble ya que no nos permitía utilizar el GoogleMaps a la ligera sin antes filtrar y utilizar una misma nomenclatura para cada calle (el nombre que le íbamos a asignar a cada calle iba a ser el que reconozca el GoogleMaps). Sorprendentemente el GoogleMaps es muy sensible con el tema de reconocer calles y por ende había muchas calles que no las reconocía por estar escrita de cierta manera. A la matriz $D$ le pasamos el siguiente algoritmo para eliminar las direcciones que no podían ser registradas.

```
c=0;
removidas=0;
posicion=zeros(3824,1);
for i=1: 3824
    for j=1: 21
        if (D(i,j)<10000)
           c=c+1;
        endif
    endfor
    if (c==0)
       D(i,:)=zeros(1,21);
       posicion(i,1)=1;
       removidas=removidas+1;
    endif
    c=0;
endfor
```

Ésto nos permitía por un lado anular ciertas cuadras que daban resultados demasiado grandes en cuestión de tiempo (lo que indica que esa direcciíon está siendo mal registrada pues está fuera de Pergamino) y registrar la cantidad de personas que estaban siendo anuladas. Luego de esto, se iba a modificar el input de forma que cada cuadra mal registrada se le asigna una cantidad de cero votantes.

Otra cuestión que también aportó al problema anterior, pero de una manera más grave, fue el error humano. Había calles que estaban escritas de una manera erronea, ya sea por error de tipeo o de ortografía o de llamar avenida a una calle que no lo era. Decimos de que aportó al problema de una manera más grave porque no permite clasificar los errores y poder armar un programa que corrija esos tipos de errores, en definitiva uno no sabe con que tipo de errores se puede encontrar.

La solución que se le dió a este problema fue que decidimos hacer dos cosas básicamente. Primero, jugar un rato en la lista de "DIRECCIONES" (que fue donde tuvimos los mayores inconvenientes) tratando de juntar de manera manual todas las direcciones que estaban escritas de forma distinta, para esto hicimos fuerte y repetido uso de la función "sort" del Excel (juntamos todas las direcciones que compartían la misma calle o nombre similar y las pusimos en subbloques dentro de la lista). Segundo, corrimos el algoritmo que descarga los archivos ".json" y nos dimos cuenta que las direcciones que el Google Maps no tomaba daba como output en sus respectivas entradas números excesivamente grandes. Esto lo utilizamos como indicadores de problemas, donde aparecía un número gigantezco había una calle que no era reconocida. Armamos una función que toma la lista que armamos de las cuadras en el paso anterior y nos indica la entrada donde no se reconoce. Luego, vimos cual de las formas en que estaba escrita la calle era la correcta y reemplazamos todas las otras formas por esa (de manera manual). Repetimos este proceso hasta que no aparecieron números muy grandes en el output del algoritmo.

En el proceso anterior había muchas calles que eran indescifrables, muchas entradas en la lista que le faltaban datos, como por ejemplo número de casa en la dirección. Por ende todas esas personas, a las cuales les faltaban datos necesarios, no las tuvimos en cuenta. Por eso es que no tomamos en cuenta el 100 % del padrón. Ésto lo tuvimos en cuenta en el momento del armado de cada lista que ha sido explicada en este capítulo.

Otra dificultad que se nos presentó fue a la hora de decidir que tipo de filas íbamos a utilizar para modelar las filas de votación. Más específicamente si las distribuciones de llegada y servicio iban a ser Markovianas, Determinísticas o Generales. Decidimos utilizar distribuciones Markovianas en ambas puesto que preferimos trabajar con un proceso estocástico sin memoria el cual la probabilidad condicional sobre el estado presente, futuro y pasado del sistema sean independientes. Sin embargo no descartamos la posibilidad de que las distribuciones de otro tipo den una mejor modelización de la realidad. Como ya espesificamos en la sección "Trabajo a Futuro" del Capítulo 1, éste es un tema que nos proponemos a analizar.

Una obsevación es que con el objetivo de hacer gráficos, hicimos un algoritmo que nos permite tomar como input una dirección y nos dá como output su latitud y longitud. ésto nos permita facilmente generar gráficos en un sistema coordenado.

Una observación importante con respecto a la solución que brinda el modelo propuesto es que se eliminaría en parte el problema del embotellamiento de automóviles el día de la elección (problema que hace perder mucho tiempo y no lo estamos teniendo en cuenta dentro del modelo) puesto que los votantes estarían yendo a votar a lugares mucho maś cercanos, cosa que evitaría el uso de automóviles en gran porcentaje de votantes.

Otra observación es que se probó varias veces el modelo, con distintos parámetros (parecidos a los reales) y los que más sensibles, en cuestione de que si se obtenía

una solución o no, eran *Equidad* y *comp*.

# Capítulo 3

# Marco Teórico

## 3.1. Teoría de Filas

### 3.1.1. Descripción de un sistema de filas

Un sistema de filas se puede describir como: clientes que llegan buscando un "servicio", esperan en éste, y abandonan el sistema una vez han sido atendidos. En algunos casos se puede admitir que los clientes abandonan el sistema si se cansan de esperar. El término cliente se usa con un sentido general y no implica que sea un ser humano, puede significar piezas esperando su turno para ser procesadas o una lista de trabajo esperando para imprimir en una impresora en red. En el caso de nuestro problema particular, sí se tratará de personas físicas.

Una representación detallada exige definir un número elevado de parámetros y funciones.

### 3.1.2. Características de los sistemas de filas

Las características básicas que se deben utilizar para describir adecuadamente un sistema de filas:
(I) Patrón de llegada de los clientes
(II) Patrón de servicio de los servidores
(III) Disciplina de fila
(IV) Capacidad del sistema
(V) Número de canales de servicio
(VI) Número de etapas de servicio

**Patrón de llegada de los clientes:**

En situaciones de fila habituales, la llegada es estocástica, es decir la llegada depende de una cierta variable aleatoria, en este caso es necesario conocer la distribución probabilística entre dos llegadas de cliente sucesivas. Adems habría que tener

en cuenta si los clientes llegan independiente o simultáneamente. También es posible que los clientes sean impacientes. Es decir, que lleguen a la fila y que tras esperar mucho rato en la fila o si es demasiada larga al llegar, decidan abandonar. Por último es posible que el patrón de llegada varíe con el tiempo. Si se mantiene constante le llamamos estacionario, si por ejemplo varía con las horas del día es no-estacionario. En el caso nuestro, modelamos con un patrón de llegada estacionario y pensamos como trabajo a futuro hacerlo no-estacionario.

**Patrones de servicio de los servidores:**

El tiempo de servicio también puede variar con el número de clientes en la fila, trabajando ms rápido o ms lento, y en este caso se llama patrones de servicio dependientes. Al igual que el patrón de llegadas, el patrón de servicio puede ser no-estacionario, variando con el tiempo.

**Disciplina de la fila:**

La disciplina de fila es la manera en que los clientes se ordenan en el momento de ser servidos de entre los de la fila. Cuando se piensa en filas de personas físicas, se admite que la disciplina de fila normal es FIFO (atender primero a quien llegó primero). Las disiplinas más usadas son las siguientes: FIFO (atender primero a quien llegó primero), $LIFO$ (atender primero al último), $RSS$ (selección aleatoria de orden de clientes), $PR$ (tener reglas de prioridades, clientes más "importantes" que otros), $GD$ (se usa esta notación para indicar que puede ser cualquiera de los casos anteriores).

En cualquier caso dos son las situaciones generales en las que trabajar. En la primera, si un cliente llega a la fila con una orden de prioridad superior al cliente que está siendo atendido, este se retira dando paso al más importante. Dos nuevos subcasos aparecen: el cliente retirado ha de volver a empezar, o el cliente retorna donde se había quedado. La segunda situación es donde el cliente con mayor prioridad espera a que acabe el que está siendo atendido.

**Capacidad del sistema:**

En algunos sistemas existe una restricción, con respecto al número de clientes que pueden esperar en la fila. A estos casos se les denomina filas finitas. Esta limitación puede ser considerada como una simplificación en la modelización de la impaciencia de los clientes.

**Número de canales del servicio:**

Cuando se habla de canales de servicio paralelos, se habla generalmente de una fila que alimenta a varios servidores mientras que el caso de filas independientes se asemeja a múltiples sistemas con sólo un servidor.

Se asume que en cualquiera de los dos casos, los mecanismos de servicio operan de manera independiente.

**Etapas de servicio:**

Un sistema de colas puede ser unietapa o multietapa. En los sistemas multietapa el cliente puede pasar por un número de etapas mayor que uno. Una fila de espera

para pagar en un supermeracado es un ejemplo de un sistema unietapa. Un ejemplo de un sistema multietapa es una tienda de ropa donde uno tiene que esperar a ser atendido primero por un vendedor, luego hacer fila para usar el probador y finalmente esperar para poder pagar. Notemos que en el último ejemplo los servidores son diferentes.

En algunos sistemas multietapa se puede admitir la vuelta atrás del cliente, esto es habitual en sistemas productivos como controles de calidad y reprocesos.

### 3.1.3. Notación básica

**Nomencaltura:**

$\lambda$= Número de llegadas por unidad de tiempo $\mu$= Número de servicios por unidad de tiempo si el servidor está ocupado

$c$= Número de servidores en paralelo

$\rho = \frac{\lambda}{c\mu}$ : Congestión de un sistema con parámetros: $(\lambda, \mu, c)$

$N(t)$: Número de clientes en el sistema en el instante t

$N_q(t)$: Número de clientes en la fila en en el instante t

$N_s(t)$: Número de clientes en servicio en el instante t

$P_n(t)$: Probabilidad que haya n clientes en el sistema en el instante $t = Pr\{N(t) = n\}$

$N$: Número de clientes en el sistema en el estado estable

$P_n$ : Probabilidad de que haya n clientes en estado estable $P_n = Pr\{N = n\}$

$L$ : Número medio de clientes en el sistema

$L_q$ : Número medio de clientes en la fila

$T_q$ : Representa el tiempo que un cliente invierte en la cola

$S$ : Representa el tiempo de servicio

$T = T_q + S$: Representa el tiempo total que un cliente invierte en el sistema

$W_q = E[Tq]$: Tiempo medio de espera de los clientes en la fila

$W = E[T]$: Tiempo medio de estancia de los clientes en el sistema

$r$: Número medio de clientes que se atienden por término medio

$P_b$: Probabilidad de que cualquier servidor esté ocupado

La notación para representar los problemas de filas que consta de hasta 5 símbolos separados por barras. $A/B/X/Y/Z$

$A$: Indica la distribución de tiempo entre llegadas consecutivas

$B$: Alude al patrón de servicio de servidores

$X$: Es el número de canales de servicio

$Y$: Es la restricción en la capacidad del sistema

$Z$: Es la disciplina de fila

**Distintos tipos de distribuciones:** De acuerdo a como vayan llegando los clientes a las filas o de acuerdo a como vayan éstos vayan siendo atendidos, podemos

asignar distintos tipos de distribución que modelen mejor la realidad. Los casos más usados son los siguientes: $M$ (distribución Exponencial o Markoviana), $D$ (distribución determinista), $E_k$ (Erlang tipo-k -k=1,2,...-), $H_k$ (mezcla de k exponenciales) $PH$ (tipo fase), $G$ (indica que se trata de cualquiera de las anteriores). No voy a profundizar en este tema, puesto que dejamos el desarrollo de una mejor modelización con respecto a este tema como trabajo a futuro. Solo voy a explicar el proceso exponencial, más adelante, que es el que usamos para nuestro modelo.

### 3.1.4. Algunos resultados generales

Se presentan en este apartado algunos resultados y relaciones para problemas $G/G/1/\infty/FIFO$ o $G/G/c/\infty/FIFO$. Osea, casos generales válidos para cualquier distribución.

**Resultados y relaciones:** Si $\rho \geq 1$ el sistema tenderá a crecer inexorablemente (estado inestable), el servicio se saturará y se producirá la acumulación de clientes en el sistema (ya sea en la fila o siendo atendidos). El número de clientes en el instante $t$, $n(t)$, es el número de llegadas que han ocurrido hasta $t$ menos el número de servicios completados hasta t. El número medio de clientes en el sistema y en la fila se puede calcular de diferentes maneras:

$$L = E[n] = \sum_{n=0}^{\infty} np_n L_q = E[n_q] = \sum_{n=c+1}^{\infty} (n-c)P_n$$

Little, en su famosa fórmula, establece una relación entre la longitud de la fila y el tiempo de espera:

$$L = \lambda W L_q = \lambda W_q$$

El tiempo de estadía de un cliente en el sistema se relaciona con el tiempo de espera de un cliente en la fila,

$$W = W_q + \frac{1}{\mu}$$

El número de clientes que por término medio se están atendiendo en cualquier momento es:

$$r = L - L_q = \lambda(W - W_q) = \frac{\lambda}{\mu}$$

En un sistema de un único servidor:

$$L - L_q = \sum_{n=1}^{\infty} np_n - \sum_{n=1}^{\infty} (n-1)p_n = \sum_{n=1}^{\infty} p_n = 1 - p_0$$

La probabilidad de que un sistema de un único servidor esté vacío es $p_0 = 1 - \rho$

La probabilidad de que un servidor (de un sistema de c servidores en paralelo) esté ocupado en el estado estable es:

$$p_b = \rho = \frac{\lambda}{c\mu}$$

El tiempo de estadía del cliente $(i+1)$ en la cola es:

$$W_q^{i+1} = \begin{cases} \mathrm{W_q^{(i)}} + \mathrm{S^{(i)}} - \mathrm{T^{(i)}}, & W_q^{(i)} + S^{(i)} - T^{(i)} > 0 \\ 0 \qquad\qquad\qquad\quad , & W_q^{(i)} + S^{(i)} - T^{(i)} \leq 0 \end{cases} \tag{3.1}$$

donde $S^{(i)}$ es el tiempo de servicio del cliente $i$, y $T^{(i)}$ es el tiempo que transcurre desde la llegada del cliente y hasta la llegada del cliente $(i+1)$

**Los procesos de Poisson y la distribución Exponencial:**

La mayor parte de los modelos de filas estocásticas asumen que el tiempo entre diferentes llegadas de clientes siguen una distribución Exponencial. O lo que es lo mismo que el ritmo de llegada sigue una distribución de Poisson. A continuación se verán las características de una distribución de Poisson y como se relacionan con la distribución Exponencial. Adoptar la distribución de Poisson implica que la probabilidad de que lleguen n clientes en un intervalo de tiempo t es:

$$p_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$$

El tiempo entre llegadas se define, de este modo, como la probabilidad de que no llegue ningún cliente:

$$p_0(t) = e^{-\lambda t}$$

siendo por tanto una distribución exponencial.

**Propiedades del Patrón de llegadas (o servicio) Poisson- Exponencial:**

El uso de este patrón de llegada (o de servicio) tiene, entre otras las siguientes propiedades:

(I) El número de llegadas en intervalos de tiempo no superpuestos es estadísticamente independiente

(II) La probabilidad de que una llegada ocurra entre el tiempo $t$ y $t+\Delta t$ es $\lambda \Delta t + o(\Delta t)$, donde $\lambda$ es la tasa de llegada y $o(\Delta t)$ cumple $\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} = 0$ De hecho $o(\Delta t)$ se podría entender como la probabilidad de que llegue más de uno.

(III) La distribución estadística del número de llegadas en intervalos de tiempo iguales es estadísticamente equivalente

$$P_n(t-s) = \frac{[\lambda(t-s)]^n}{n!} e^{-\lambda(t-s)} \qquad \forall t, s \geq 0, \quad t > s$$

(IV) Si el número de llegadas sigue una distribución de Poisson el tiempo entre llegadas sigue una distribución exponencial de media $(1/\lambda)$ y al contrario

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \quad \Leftrightarrow \quad P_0(t) = e^{-\lambda t}$$

(V) Si el proceso de llegada es Poisson, los tiempos de llegada son completamente aleatorios con una función de probabilidad uniforme sobre el periodo analizado.

$f_\tau(t_1, t_2, ..., t_k/k \text{ llegadas en } [0,T]) = \frac{k!}{T^k}$

(VI) Para conocer los datos que definen un proceso de Poisson solo es necesario conocer el nmero medio de llegadas

(VII) Amnesia de la Distribución exponencial: La probabilidad de que falten $t$ unidades para que llegue el siguiente cliente es independiente de cuanto tiempo llevamos sin que llegue ningún cliente.

$$P_r\{T \leq 1 | T \geq t_0\} = P_r\{0 \leq T \leq t_1 - t_0\}$$

### 3.1.5. El sistema $M/M/1/\infty/FIFO$

Una fila $M/M/1/\infty/FIFO$ tiene un único servidor y las tasas de llegada y de servicio siguen una distribución de Poisson, siendo por tanto:

La tasa de llegada es $a(t) = \lambda e^{-\lambda t}$

La tasa de salida es $a(t) = \mu e^{\mu t}$

La probabilidad de que haya n clientes es:

$$p_n = (1-\rho)\rho^n \quad con \quad \rho = \frac{\lambda}{\mu}$$

El número medio de clientes en la fila es:

$$L = E[n] = \sum_{n=0}^{\infty} n p_n = (1-\rho)\sum_{n=0}^{\infty} n\rho^n = (1-\rho)\rho\sum_{n=0}^{\infty} n\rho^{n-1}$$

Como

$$\sum_{n=0}^{\infty} n\rho^{n-1} = \frac{\partial(\sum n\rho^n)}{\partial\rho} = \frac{\partial(\frac{1}{1-\rho})}{\partial\rho} = \frac{1}{(1-\rho)^2}$$

De donde

$$L = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda}$$

Y aplicando las relaciones fundamentales antes vistas

$$W = \frac{L}{\lambda} = \frac{1}{\mu-\lambda} L_q = \frac{\lambda^2}{\mu(\mu-\lambda)} W_q = \frac{\rho}{\mu-\lambda}$$

La cola media cuando el sistema no está vacío es:

$$L'_q = \frac{\mu}{\mu-\lambda}$$

Otro resultado interesante es conocer cual es la probabilidad de que haya $X$ o ms elementos en el sistema.

$$P(n \geq X) = \rho^X$$

## 3.2. Programación Lineal y Lineal Entera (PLE)

### 3.2.1. Programación Lineal:

La Programación Lineal en escencia trata de maximizar y/o minimizar una función lineal de dos o más variables teniendo en cuenta que las mismas deben cumplir determinadas exigencias de acuerdo a lo que querámos modelar [4] .

En un problema de programación lineal de muchas variables $x_i$, se trata de optimizar (hacer máxima o mínima, según los casos) una función (llamada *función objetivo*) de la forma:

$$Z \quad = \sum_{i=1}^{n} x_i$$

Sujeta a una serie de restricciones dadas mediante un sistema de inecuaciones lineales del tipo:

$$\sum_{i=1}^{n} x_i \leq c_j \quad , \quad \forall j = 1, ..., m$$

Los puntos del plano que cumplen el sistema de desigualdades forman un recinto convexo acotado (poligonal) o no acotado, llamado *región factible* del problema.

Todos los puntos de dicha región cumplen el sistema de desigualdades. Se trata de buscar, entre todos esos puntos, aquel o aquellos que hagan el valor de $Z$ máximo o mínimo, según sea el problema.

Los puntos de la región factible se denominan *soluciones factibles* . De todas esas soluciones factibles, aquellas que hacen óptima (máxima o mínima) función objetivo se llaman *soluciones óptimas* .

En general,un problema de programación lineal puede tener una, infinitas o ninguna solución. Lo que si se verifica es la siguiente propiedad:

**Propiedad:**

Si hay una única solución óptima, ésta se encuentra en un vértice de la región factible, y si hay infinitas soluciones óptimas, se encontrarán en un lado de la región factible.

Es posible que no haya solución óptima, pues cuando el recinto es no acotado, la función objetivo puede crecer o decrecer indefinidamente.

La región factible puede estar acotada o no.

### 3.2.2. Programación Lineal Entera:

Con el término Programación lineal entera, (PLE), nos referiremos a problemas que formalmente son problemas de programación lineal, Max ó Min $Z = Ax =$

$b$, $x \geq 0$, pero en los que algunas variables están restringidas a tomar valores enteros.

Por ejemplo, $x_1 \geq 0; x_2 \geq 0$ y entera, $x_3 \epsilon \{0, 1\}$, $x_1$ una variable como las que hemos manejado hasta ahora, $x_2$ una variable entera no negativa y $x_3$ una variable binaria, que toma únicamente dos valores, 0 ó 1.

Las variables binarias $x_j \epsilon \{0, 1\}$, pueden utilizarse para modelar situaciones en las que se decide si una acción se realiza, $x_j = 1$, o si no se realiza, $x_j = 0$.

Como veremos en la siguiente sección los existen ciertos problemas de programación lineal entera que nos van a permitir modelar las situaciones que deseamos para la resolución del problema planteado en el Capítulo 1, cosa que no podríamos hacer solo con programación lineal. Pero a cambio la resolución de los problemas será mucho más costosa, presenterán, en general, un costo computacional mucho más elevado que el de la programación lineal. La causa de este incremento de costo computacional se debe a que se pierde la deseable propiedad existente en los problemas de programación lineal de que al menos una solución óptima del problema se encuentra en un punto extremo.

En estos problemas los conjuntos ya no tienen que ser conexos (pueden estar definidos a trozos) y mucho menos convexos con lo que la idea de punto extremo tal y como la hemos definido desaparece. De todos modos, para su resolución aún podremos utilizar técnicas basadas en el simplex. Para obtener una explicación más detallada de Programación lineal entera buscar en [7].

## 3.3. Problemas de Asignación y de la Mochila

Para generar el modelo de optimización que representaba al sistema de distribución de votantes en una elección de tal manera que respete con lo planteado en el Capítulo 1, lo que hicimos fue usar dos tipos de modelos de optimización lineal entera clásicos existentes, combinarlos y complejizarlos para que se adecúe bien a lo que queríamos modelar.

Estos modelos existentes son los que resuelven los problemas de asignación y los problemas del tipo Mochila.

**Problema de Asignación:** Se trata de asignar la realización de n tareas a m personas (máquinas, etc.). Este problema es un caso particular de otro problema clásico de optimización, el problema de transporte.

El problema consiste en minimizar el coste total de realizar las tareas sabiendo que cada tarea $i$ debe ser hecha por una única persona y cada persona $j$ debe realizar una única tarea, siendo $c_{ij}$ el coste de realizar la tarea $i$ por la persona $j$, para profundizar en el tema buscar en [1].

Una modificación del problema sería que no necesariamente cada tarea tenga que ser hecha por solo una persona, sino que pueda que sea realizada por más personas.

En este caso se tendría la minimización del coste de las tareas que podrían encargarse un conjunto de personas.

Las variables del problema

$$x_{ij} = \begin{cases} 1, & \text{se asigna la tarea } i \text{ a la persona } j \\ 0, & \text{c.c.} \end{cases}$$

$$Minimizar \quad Z = \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

$$\sum_{i=1}^{n} x_{ij} \quad , \quad \forall i = 1, ..., n$$

$$\sum_{j=1}^{m} x_{ij} \quad , \quad \forall j = 1, ..., m$$

$$x_{ij} \geq 0$$

**Problema de la Mochila (Knapsack):** Se trata de maximizar el valor total de la elección de un conjunto de n proyectos sin sobrepasar el presupuesto $b$ disponible, siendo $v_j$ y $c_j$ el valor y coste de cada proyecto $j$ respectivamente. El nombre procede de la decisión que toma un montañero que trata de maximizar el valor de lo que introduce en su mochila con una restricción de máximo peso admisible [8]. Las variables del problema son

$$x_{ij} = \begin{cases} 1, & \text{se elije el proyecto } j \\ 0, & \text{c.c.} \end{cases}$$

Ésta es una utilización habitual de las variables binarias como forma de seleccionar una alternativa, un proyecto en este caso. La formulación del problema es la siguiente

$$Maximizar \quad Z = \sum_{j=1}^{n} v_j x_j$$

$$\sum_{j=1}^{n} c_j x_j \leq b$$

$$x_j \epsilon \{0, 1\}$$

## 3.4. Métodos de Resolución de Problemas de PLE

Propondremos una introduccin a los principales algoritmos para obtener la solución a este tipo de problemas. En concreto, se estudian los métodos siguientes:

(I) **Método de bifurcación y acotación:** La solución al problema de programación lineal entera ($PPLE$) original se obtiene resolviendo una secuencia ordenada de problemas de programación lineales ($PPLs$) que se obtienen relajando las restricciones de integralidad y añadiendo restricciones adicionales.

(II) **Método de los planos de corte:** En esta técnica, se resuelve el problema original relajado en el que se incluyen restricciones adicionales, denominadas *cortes de Gomoroy* , que reducen la región factible sin excluir soluciones que cumplen las condiciones de integralidad.

### 3.4.1. Método de bifurcación y acotación para un PPLE Mixta:

El método de bifurcación y acotación ( $B\&B$ , de Branch and Bound) resuelve un $PPLE$ resolviendo una secuencia ordenada de $PPLs$ que se obtienen relajando las restricciones de integralidad y aadiendo restricciones adicionales. El nmero de restricciones adicionales crece a medida que el mtodo $B\&B$ progresa. Estas restricciones permiten separar la regin factible en subregiones complementarias.

El método $B\&B$ establece inicialmente cotas inferior y superior del valor óptimo de la función objetivo. El mecanismo de bifurcación aumenta progresivamente el valor de la cota inferior y disminuye también progresivamente el valor de la cota superior. La diferencia entre estas cotas es una medida de la proximidad de la solución actual a la óptima, si ésta existe.

Al minimizar, se obtiene una cota inferior de la solución óptima relajando las restricciones de integralidad del $PPLE$ inicial y resolviendo el $PPL$ resultante. Además, el valor de la funcin objetivo para cualquier solución del $PPLE$ original es una cota superior de la solución óptima. De manera análoga, al maximizar, la solución del $PPL$ relajado es una cota superior para el óptimo y cualquier solución del $PPLE$ original es una cota inferior de la solución óptima.

A continuación se enumeran los pasos del algoritmo $B\&B$ para un $PPLE$ Mixta:

**Paso 1: Iniciación**

(I) Se establece una cota superior ($\infty$) y una cota inferior (-$\infty$) de la solución óptima. (II) Se resuelve el $PPLE$ Mixta (algunas variables son enteras y las restantes son continuas) inicial relajando las restricciones de integralidad. (II.a) Si el problema relajado es infactible, el original tambin lo es y no hay solución. (II.b) Si la solución obtenida satisface las condiciones de integralidad, es óptima. (II.c) En cualquier otro caso, se actualiza el valor de la cota correspondiente con el valor de la función objetivo resultante.

**Paso 2: Bifurcación**

(I) Empleando la variable $x_k$ que ha de ser entera y no lo es, se generan mediante bifurcación dos problemas. Si el valor de la variable que ha de ser entera $x_k$ es $a.b$, donde $a$ y $b$ son sus partes entera y fraccional respectivamente, los problemas fruto de la bifurcación son los siguientes.

(I.a) El primer problema es el $PPLE$ relajado al que se la ańade la restriccin $x_k \leq a$.

(I.b) El segundo es el $PPLE$ relajado al que se le añade la restricción $x_k \leq a+1$.

(II) Estos problemas se colocan ordenadamente en una lista de problemas a procesar que son resueltos secuencialmente o en paralelo. Obsérvese que la técnica de bifurcación propuesta cubre completamente el espacio de soluciones.

**Paso 3: Solución**

(V) Se resuelve el siguiente problema en la lista de problemas a procesar.

**Paso 4: Acotación**

(VI) Si la solución del problema actual satisface las condiciones de integralidad y el valor óptimo de su función objetivo es menor que la cota superior actual, dicha cota se actualiza al valor óptimo de la función objetivo del problema resuelto, y el minimizador actual se almacena como el mejor candidato a minimizador del problema original. En caso de maximizaciones, la cota inferior actual se actualiza al valor óptimo de la función objetivo del problema resuelto si éste es menor que dicha cota inferior.

(VII) Si, por el contrario, la solución obtenida no satisface las restricciones de integralidad y el valor de la correspondiente función objetivo esta entre las cotas inferior y superior, se actualiza el valor de la cota inferior al valor de la funcin objetivo del problema resuelto y se procede a bifurcar de nuevo. En caso de maximizaciones, se actualiza el valor de la cota superior al valor de la función objetivo del problema resuelto y se procede a bifurcar de nuevo.

(VIII) Los problemas generados en el proceso de bifurcación se añaden a la lista de problemas que han de resolverse.

**Paso 5: Poda**

(IX) Poda por cotas: Tiene lugar si la solución no satisface las condiciones de integralidad y además el valor de la funcin objetivo del problema resuelto es mayor que la cota superior para minimizaciones o menor que la cota inferior para maximizaciones. En este caso no es posible obtener soluciones mediante bifurcaciones adicionales de esa rama.

(X) Poda por infactibilidad: Tiene lugar si el problema es infactible.

(XI) Poda por integralidad: Tiene lugar si la solución del problema actual cumple las restricciones de integralidad.

**Paso 6: Optimalidad**

(XII) Si la lista de problemas a procesar no está vacía, se continua con el Paso 3.

(XIII) Si la lista de problemas a procesar está vacía, el procedimiento concluye.
(XIV) Concluido el problema, si existe un candidato a minimizador, dicho candidato es el minimizador; en caso contrario, el problema es infactible.

El algoritmo de $B\&B$ devuelve la solución óptima o notifica la infactibilidad bien en el Paso 1 ó en el paso 6. El proceso de bifurcación concluye por la poda de la rama correspondiente como consecuencia de una de las tres razones siguientes:
1. La solución del problema relajado es mayor que la cota superior disponible en el caso de minimizaciones, o menor que la cota inferior disponible para el caso de maximizaciones.
2. El problema considerado es infactible.
3. La solución obtenida satisface las condiciones de integralidad.
Como puede verse, los pasos centrales del algoritmo $B\&B$ son la bifurcación, la acotación y la poda. La diferencia entre un algoritmo $B\&B$ u otro radica en las diferentes estrategias que pueden llevarse a cabo a la hora de implementar tales pasos.

### 3.4.2. Estrategias de bifurcación y procesamiento

Cualquier variable que deba ser entera pero que no lo sea en la solución actual, es una variable candidata para bifurcación. Cuál escoger no es una cuestión trivial, y su respuesta ha de basarse en la estructura del problema.

Los problemas almacenados para ser procesados pueden tratarse mediante estrategias en profundidad, en anchura o mixtas. La siguiente figura ilustra las dos primeras alternativas. Normalmente el conocimiento técnico del problema permite establecer el tipo de estrategia a utilizar.

(I) Una estrategia de procesado en profundidad origina rápidamente problemas fuertemente restringidos que producen buenas cotas superiores e inferiores. Da lugar asimismo a problemas infactibles y por tanto a una deseable eliminación de ramas.

(II) Por el contrario, una estrategia en anchura permite tratar problemas muy similares, de lo que pueden desprenderse ventajas computacionales como es la reoptimización eficiente del problema relajado actual partiendo de la solución del anterior.

## 3.5. Algoritmo de los cortes de *Gomoroy* para un PPLE

A continuación se enumeran los pasos del algoritmo de los cortes de *Gomoroy* para un PPLE:

**Paso 1: Iniciación**

(I) Se resuelve el problema original sin restricciones de integralidad.

(I.a) Si la solución no está acotada, el problema original no está acotado y se para.

(I.b) Si el problema es infactible, el problema original también lo es y se para.

(I.c) En cualquier otro caso, se continua con el paso siguiente.

**Paso 2: Control de optimalidad**

(II) Si la solución obtenida cumple las condiciones de integralidad, se para dado que esta solución es óptima.

(III) En cualquier otro caso, se continua con el paso siguiente.

**Paso 3: Generación de un corte**

(IV) Se emplea una variable básica que ha de ser entera pero no lo es para generar un corte de *Gomory* .

**Paso 4: Resolución**

(V) Se añade el corte de *Gomory* obtenido al problema previamente resuelto, se resuelve el problema resultante y se continúa con el paso 2.

Deben además tenerse en cuenta los aspectos siguientes:

- Obsérvese que el número de restricciones crece con el número de iteraciones.
- Puesto que en cada iteración se añade una nueva restricción, debe emplearse para la resolución del problema el método simplex dual. Esto es así puesto que al añadir una nueva restricción, el problema primal correspondiente se hace infactible pero su dual permanece factible, aunque no óptimo.
- Por tanto, para resolver el nuevo problema puede partirse de la solución dual del problema previo (sin restricción adicional), lo que supone una ventaja computacional importante.

# Capítulo 4

# Conclusión

Como conclusión de este trabajo decimos que se pudo lograr los objetivos principales del mismo los cuales eran: Lograr dar un ejemplo de un proceso real que puede ser drásticamente mejorado con la utilización de herramientas computacionales y de investigación operativa, generar un modelo de asignación de votantes a sus lugares de votación de manera que podamos minimizar los tiempos promedio de votación.

El modelo que se logró construir proporcionó una solución al modelo que disminuye el tiempo promedio de votación a menos de la mitad, un resultado que hace ver el poder de la investigación operativa. Además, se dejó la estructura básica para profundizar el modelo y tener así una mejor representación de la realidad.

Es indiscutible el mejoramiento que trae en procesos de cualquier índole las herramientas computacionales. Sin embargo, no se aprovechan lo que se deberían y por eso es necesario generar trabajos como éste que muestren su efectividad. Para eso lo mejor es una vez terminado el trabajo poder disponer de las herramientas institucionales como para poder aplicarlo en caso de ser posible.

Tuvimos una experiencia directa con la manipulación y armado de bases de datos, cosa que nos permitió ver la dificultad de este paso si no se tiene en cuenta a priori que en algún futuro esas bases de datos serán utilizadas dentro de un modelo computacional. Por eso recomendamos fuertemente a todas las personas que estén en proyectos donde se generen bases de datos, que se tenga en cuenta la posibilidad de que éstas sean utilizadas dentro de un modelo. Las cosas que hay que tener en cuenta son muy sencillas de preveer, se necesita tomar estándares comunes y prolijidad.

El fomentar este tipo de herramientas es indispensable para el desarrollo del país puesto que estamos entrando en la era del conocimiento, donde ya tenemos grandes cantidades de bases de datos y lo que hace falta es poder interpretarlas para la toma de decisiones. El cambio del sistema científico nacional debe seguir siendo profundizado para dar lugar a una ciencia que esté al servicio de la sociedad, resolviendo sus problemas y necesidades.

## 4.1. Agradecimientos

Agradezco especialmente a mis directores de tesis Guillermo A. Durán y Nicolás E. Stier-Moses que me guiaron en todo el proceso, enseñaron a como encarar un problema aplicado y como uno debe comunicarlo.

Tambin agradezco a mis profesores Damián Fernandez F. y Cristina Turner que me ayudaron mucho desde Córdoba.

A mi familia que dedicó mucho esfuerzo en acompañarme.

A Nirvana B. Caballero por todo.

Agradezco a mis amigos.

Agradezco al Estado que permite una universidad pública, gratuita, laica y de calidad.

Y finalmente agradezco a la Presidenta de la Nación Argentina Cristina Fernandez de Kirchner por apostar al desarrollo del país, sobre todo en materia de ciencia.

Muchas gracias a todos ellos.

# Bibliografía


[1] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment problems*. Cambridge University Press, 2012.

[2] I. Hui Cain B. and K. MacDonald. *Sorting or Self-Sorting? Competition and Redistricting in California*. The New Political Geography of California, Berkeley Public Policy Press, 2008.

[3] IBM ILOG CPLEX. www.cplex.com. 2013.

[4] A. Herrán. Introduccion a la programación matemática. 2009.

[5] Google Inc. The google geocoding api. 2013.

[6] José Pedro García Sabater. Teoría de colas. 2011.

[7] Alexander Schrijver. *Theory of linear and integer programming*. Wiley, 1998.

[8] Begoña Vitoriano. Modelos operativos de gestión. 2009.

[9] Wikipedia. Gerrymandering. 2013.