ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ciencias Naturales y Matemáticas

Análisis experimental del problema del camino más corto con restricciones en los nodos de un grafo.

PROYECTO INTEGRADOR

Previo la obtención del Título de:

Ingeniero en Logística y Transporte

Presentado por:

Joddel Bryan Varas La Rosa

Héctor Fernando Alvarado Ostaíza

GUAYAQUIL - ECUADOR

Año: 2022

ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

College of Natural Sciences and Mathematics

Análisis experimental del problema del camino más corto con restricciones en los nodos de un grafo.

CAPSTONE COURSE

A project submitted in partial fulfillment of the requirements for the degree of:

Logistics and transportation Engineer

By:

Joddel Bryan Varas La Rosa

Héctor Fernando Alvarado Ostaíza

GUAYAQUIL - ECUADOR

2022

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; *Joddel Bryan Varas La Rosa y Héctor Fernando Alvarado Ostaíza* y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Joddel Varas

Héctor Alvarado

EVALUADORES

Firmado electrónicamente por:
DAVID ANTONIO DE SANTIS BERMEO

David De Santis

PROFESOR DE LA MATERIA

Firmado electrónicamente por:
JOSE XAVIER CABEZAS GARCIA

Xavier Cabezas

PROFESOR TUTOR

ÍNDICE GENERAL

AGRADECIMIENTOS

Llenos de una profunda sinceridad y amor dedicamos este agradecimiento a la ESPOL por ser el lugar donde se nos otorgó todo los conocimientos invaluables y experiencias memorables en toda nuestra vida académica. Al esfuerzo constante de cada uno de sus

profesores que nos convirtieron cada día en un mejor profesional, en especial a aquellos que consideraron dar un poco más, nos alentaron y fueron parte de este camino en cuál tomaron la responsabilidad de ser nuestro guía. A nuestro tutor Xavier Cabezas, Ph.D por la sabiduría, la paciencia, y la oportunidad de aprender más. A la facultad FCNM por permitirnos adquirir conocimientos y experiencia en nuestro campo de estudio.

RESUMEN

En la actualidad existe un gran interés en el estudio de las generalizaciones de los algoritmos de búsqueda, esto debido a la variedad de problemáticas que se buscan resolver. Uno de los principales algoritmos que se considera para estas generalizaciones es el del camino más corto (SPP), cómo lo son los algoritmos de camino más corto con ventanas tiempo (TCSPP). El presente proyecto tuvo por objeto desarrollar un análisis experimental de una nueva variante del problema del camino más corto, la cual consiste en agregar restricciones de semaforización a los nodos de una red con el objetivo de considerar el tiempo de espera en una intersección por cambio de luces de los semáforos.

Para este análisis, se planteó un modelo matemático de este nuevo problema, se diseñaron dos pseudócodigos de programación de algoritmos modificados para luego

proceder a diseñar un código de programación en Python para cada pseudocódigo y, por último, se modelizó en GAMS un modelo de programación lineal (LP) del problema.

El resultado del análisis detalla las diferencias que existen en resolver este problema entre dos modificaciones de algoritmos conocidos. La comparación de resultados de una instancia creada, obtenidos mediante un modelo de programación lineal contra los métodos heurísticos.

La validación de la importancia del aporte de este proyecto se ve refleja en todo el análisis de resolver el problema mediante dos algoritmos ya conocidos, la fomentación a nuevas consideraciones del problema y la implementación del uso de ventanas de tiempo para resolver el problema.

Palabras clave: Análisis, Variante, SPP, Heurísticas, Programación lineal.

ABSTRACT

Currently, there is great interest in the study of generalizations of search algorithms, due to the variety of problems they seek to solve. One of the main algorithms considered for these generalizations is the shortest path algorithm (SPP), as well as the time-windowed shortest path algorithms (TCSPP). The objective of this project was to develop an experimental analysis of a new variant of the shortest path problem, which consists of adding traffic light constraints to the nodes of a network to consider the waiting time at an intersection due to a change of direction. traffic lights.

For this analysis, a mathematical model of this new problem was proposed, two modified algorithm programming pseudocodes were designed to then proceed to design a Python programming code for each pseudocode and, finally, a programming model was modeled in GAMS. linear (LP) of the problem.

The result of the analysis details the differences that exist in solving this problem between two modifications of known algorithms. The comparison of results of a created instance, obtained through a linear programming model against heuristic methods.

The validation of the importance of the contribution of this project is reflected in the entire analysis of the solution of the problem through two already known algorithms, the promotion of new considerations of the problem and the implementation of the use of time windows to solve the problem.

***Keywords**: Analysis, Variant, SPP, Heuristics, Linear Programming.*

ABREVIATURAS

SPP = Shortest Path Problem

TCSPP = The shortest path problem with time constraints

NP= Polinimial No Deterministico

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

ANTECEDENTES

Uno de los artículos tomados en consideración es el de (Yen-Liang & Yang, 2000), quien menciona sobre nuestro tema de proyecto como: “El problema del camino más corto con restricciones de tiempo (TCSPP) es una generalización importante del camino más corto problema (SPP) y ha atraído un amplio interés de investigación en los últimos años…Este artículo presenta una novela restricción de tiempo, llamada restricción de semáforo, para simular las operaciones de control de semáforo encontradas en intersecciones de caminos. Básicamente, la restricción consiste en una secuencia repetida de ventanas de tiempo. En cada ventana, solo los autos en rutas especificadas pueden pasar a través de la intersección”.

Este artículo esta estrictamente relacionado con nuestro tema de proyecto y es uno de los principales, aquí nos muestra cómo se realiza una modelación el control de los semáforos que intervienen en una red uno de los temas importantes que se debe de definir en este proyecto. Además, nos proporciona un método de solución que permite resolver el problema de encontrar el camino más corto en una red establecida con semaforización y un pseudocódigo".

El problema del camino más corto (SPP) se concentra en encontrar el camino con la distancia mínima, tiempo o costo desde un origen hasta un destino a través de una red conectada. Es un clásico y problema importante en el área de optimización combinatoria debido a sus numerosas aplicaciones y generalizaciones en comunicaciones y redes de transporte. Se han publicado varias reseñas excelentes sobre el SPP. (Jiang Jin & Chen, 2013) Es otro de los artículos que muestra una solución donde se habla del algoritmo modificado Dijkstra.

Este artículo nos proporciona definiciones de literatura relacionada con el problema del camino más corto, ventanas de tiempo y la problemática en sí. Además, nos muestra cómo se genera una red con restricciones en los nodos, definiendo las variables de decisión para consiguiente obtener K caminos que óptimos de llegar desde un punto a otro en la red. Por último, luego de obtener las K soluciones óptimas, se aplica un algoritmo modificado de Dijkstra, que proporciona una solución óptima.

1. INTRODUCCIÓN

Las decisiones que actualmente enfrentan ciertos sectores de la sociedad al momento de determinar que rutas deben de considerar para la movilización de entes en una red establecida, es considerable. En el sector empresarial, principalmente en aquellas empresas que se dan servicio de transportación, es necesario conocer cómo seleccionar la ruta que minimice el tiempo, distancia recorrida o maximice el valor económico comercial.

En la actualidad existen una gran variedad de investigaciones que tratan de resolver las distintas situaciones que se pueden dar en un sistema o red de transporte. Se sabe que,

en una situación real, ir de un punto de origen a un punto destino se debe de tomar en cuenta ciertas limitaciones y restricciones que intervienen en la búsqueda de una solución óptima, y son justamente estas variantes y la cantidad de estudios que se han hecho sobre este tipo de problemas, lo que permiten tener una amplia cantidad de algoritmos, modelos o distintas metodologías que resuelven.

El consiguiente trabajo se basa en una variable de un problema ya conocido, el cuál es el del camino más corto. Hace énfasis en la intervención de restricciones de semaforización que se da al momento de efectuar una búsqueda de una solución en una red que representa un sistema vial o de transporte. Aquí se busca plantear esta problemática tomando en cuenta las distintas variables y restricciones a considerar para así llevar a cabo un modelo matemático para este problema. Se tomará en cuenta varios algoritmos de solución al problema del camino más corto como lo son el Algoritmo de Dijkstra, Bellman Ford y Floyd Warshall para la implementación de variantes de estos algoritmos.

Se implementará un código de programación para los algoritmos modificados de Dijkstra y Bellman Ford, con estos se llevará a cabo la ejecución de distintos escenarios para evaluar las soluciones obtenidas. Por otro lado, tomando el modelo matemático planteado al inicio de nuestro proyecto se efectuará un modelo de programación lineal.

1.1 Descripción del problema

La problemática por resolver es encontrar la ruta con el valor mínimo de tiempo transcurrido de una red, partiendo desde un punto de origen y llegando a uno de destino, ambos establecidos, tomando en cuenta que existen restricciones de semaforización en los puntos que forman parte de esta red os sistema.

Nuestra problemática tiene relación con otra, conocida en temas de estudios como el problema del camino más corto, este, es denominado uno de los más importante de optimización y combinatoria, debido a la variedad de aplicaciones que puede tener y a los diferentes estudios que se han realizado tomando en cuenta este problema en común, ya sea con una nueva variante del problema o profundizando más en los algoritmos ya conocidos que lo resuelven.

Es por lo antes mencionado, la importancia que comprende nuestro tema a resolver, porque es una variante de un problema de gran relevancia y como variante de este, se lo puede catalogar de mucha utilidad para diferentes áreas de estudio.

1.2 Objetivos

1.2.1 Objetivo General

Analizar experimentalmente el problema de la ruta más corta con restricciones en los nodos mediante un algoritmo de ruta más corta, utilizando diferentes métodos de optimización para determinar un camino óptimo

1.2.2 Objetivos Específicos

- Formular un modelo matemático para el problema de la ruta más corta con restricciones en los nodos.
- Programar una heurística del camino más corto considerando variante de semaforización
- Comparar los resultados y hacer un estudio con relación a sus variantes
- Resolver el problema mediante un modelo de programación lineal y comparar los resultados con los métodos heurísticos

1.3 Marco Teórico

1.3.1 Grafos

El origen del vocablo grafo procede de la antigua Grecia, en donde se hacía uso de este término para referenciar a dibujos o imágenes. Un grafo es la forma de representar un conjunto no vacío, en donde sus elementos que lo conforman tienen una relación y son llamados comúnmente cómo vértices y aristas.

• Vértices: También denominados nodos. Es una unidad principal del grafo, se los simbolizan de manera gráfica como un punto y este puede simbolizar diferentes elementos dependiendo el análisis de estudio.

• Aristas: La forma gráfica de representarlas en grafo es mediante líneas. Estos están conectados a dos nodos y expresan una relación entre estos, donde este enlace también dependerá del estudio realizado mediante el grafo.

En la actualidad existe una gran variedad de aplicaciones para los grafos, esto debido a que permiten representar una diversidad de situaciones en distintas áreas de estudios.

Por ejemplo: circuitos eléctricos, planos de sistemas de transporte o carreteras, metodologías sobre evaluación programas, representaciones y análisis de redes informáticas, entre otros.

1.3.1.1 Definición de un Grafo

Se denomina un Grafo al par (V, E), dónde V es un conjunto que posee elementos llamados vértices y nodos, mientras que E es un multiconjunto conformado por pares no ordenados de vértices llamados aristas o arcos. Si se tiene la notación de V cómo $V = \{v_1, v_2, v_1, \ldots, v_n\}$, entonces los elementos del multiconjunto E se expresa de la siguiente manera $\{v_i, v_j\}$ donde $i \neq j$ y estos elementos a_i y a_j se le llaman extremos de $\{v_i, v_j\}$. Si $\{v_i, v_j\} \in E$, entonces podemos decir que v_i es adyacente v_j y viceversa.

1.3.1.2 Grafo simple

Un grafo se denomina simple si como máximo existe una arista que conecta a dos vértices cuales quiera.

1.3.1.2.1 Definición de un grafo simple

Sea un grafo $G(V, E)$, dónde V posee elementos llamados vértices y nodos, mientras que E está conformado por pares no ordenados de vértices distintos elementos de V llamados aristas o arcos.

1.3.1.3 Grafo dirigido

Se denomina un grafo dirigido o dígrafo, al que posee aristas con un sentido o dirección común.

1.3.1.3.1 Definición de grafo dirigido

Sea un grafo $G(V, E)$, dónde V posee elementos llamados vértices y nodos, mientras que E está conformado por pares ordenados de vértices elementos llamados aristas o arcos V.

1.3.1.4 Grafo ponderado

Un grafo tiene ponderación o es ponderado si se le asigna un valor a cada una de las aristas que relaciona dos nodos.

1.3.1.4.1 Definición de grafo ponderado

Sea un Grafo (V, E)*, dónde* V *posee elementos llamados vértices y nodos, mientras que* E *conformado por pares no ordenados de vértices llamados aristas o arcos y se tiene la notación de* V *cómo* $V = \{v_1, v_2, v_1, \ldots., v_n\}$*, entonces los elementos del multiconjunto* E *se expresa de la siguiente manera* $\{v_i, v_j\}$*. Además, existe un valor* $w(i, j)$ *asociado a cada una de las aristas del grafo que relaciona a las aristas* a_i *y* a_j*.*

1.3.1.5 Grafo convexo

Un grafo es convexo si para cada uno de los vértices existe a al menos una arista que lo relacione con otro vértice.

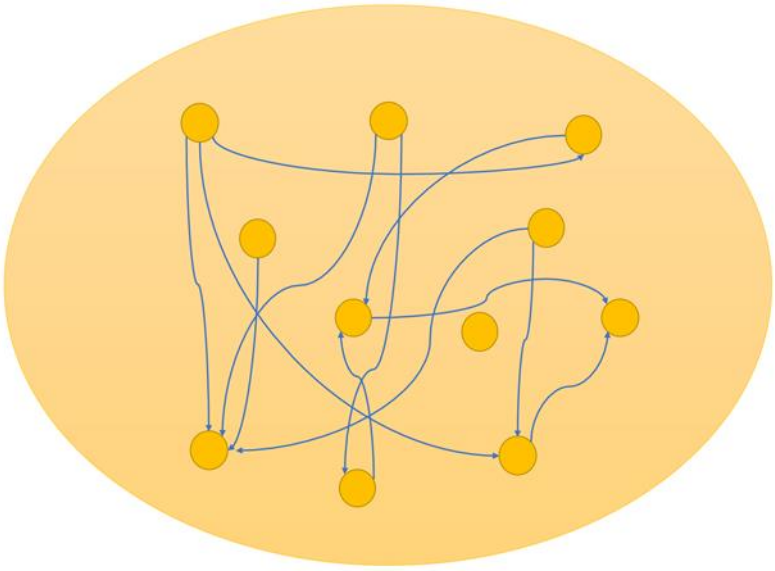
1.3.1.5.1 Definición de un grafo convexo

Sea un Grafo (V, E)*, dónde* V *posee elementos llamados vértices y nodos, mientras que* E *conformado por pares no ordenados de vértices llamados aristas o arcos y se tiene la notación de* V *cómo* $V = \{v_1, v_2, v_1, \ldots., v_n\}$*, entonces los elementos del multiconjunto* E *se expresa de la siguiente manera* $\{v_i, v_j\}$ *donde para cada* i *un* j*.*

1.3.1.6 Red de un grafo

Es la forma de representar la relación o vinculo que tiene un finito puntuales elementos con otros, donde estos forman parte de un determinado conjunto. Una red se utiliza para mostrar distintas situaciones de un determinado evento natural donde existe una relación entre sus actores, por ejemplo, una red de transporte donde los elementos que forman

parte de esta red pueden ser terminales, paradas o en general determinadas ubicaciones y la relación de accesibilidad, conexión o distancia que existe entre estas.



1 *Figura 1. 1. Representación de una Red de un grafo. Fuente: Varas & Alvarado*

1.3.1.7 Árbol generador

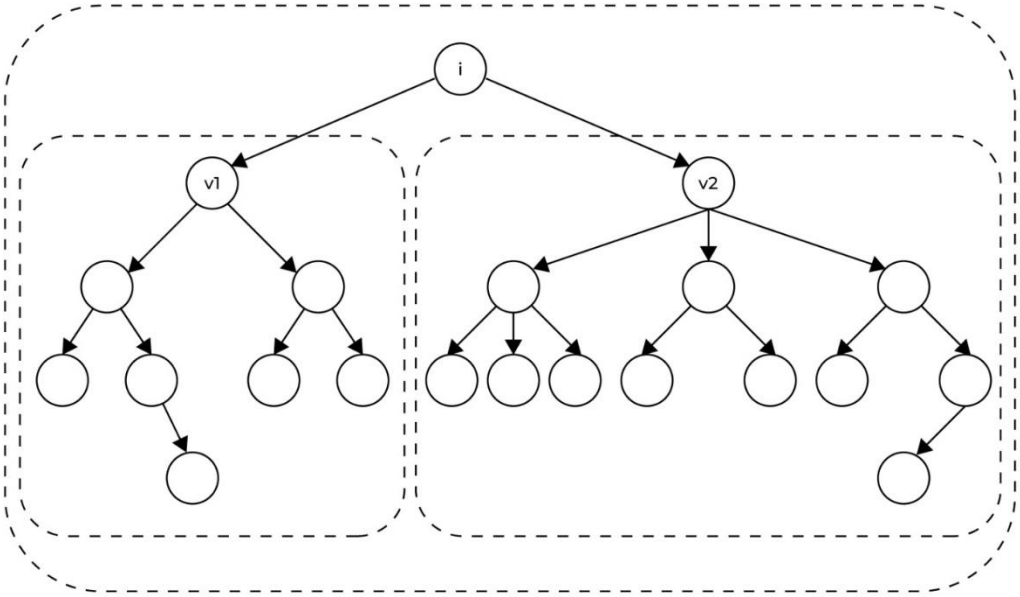
El termino árbol se refiere para a expresar de manera gráfica a grafos que no forman ciclos en forma de ramificación.

1.3.1.7.1 Propiedades

- Todo árbol con al menos dos vértices tiene al menos dos hojas.
- Si borramos una hoja de un árbol obtenemos otro árbol.

1.3.1.7.2 Definición de un árbol generador.

Según (Safe, 2018) *un árbol generador de una gráfica G es una subgráfica* T *de* G *tal que* T *es un árbol, y* $V(T) = V(G)$.



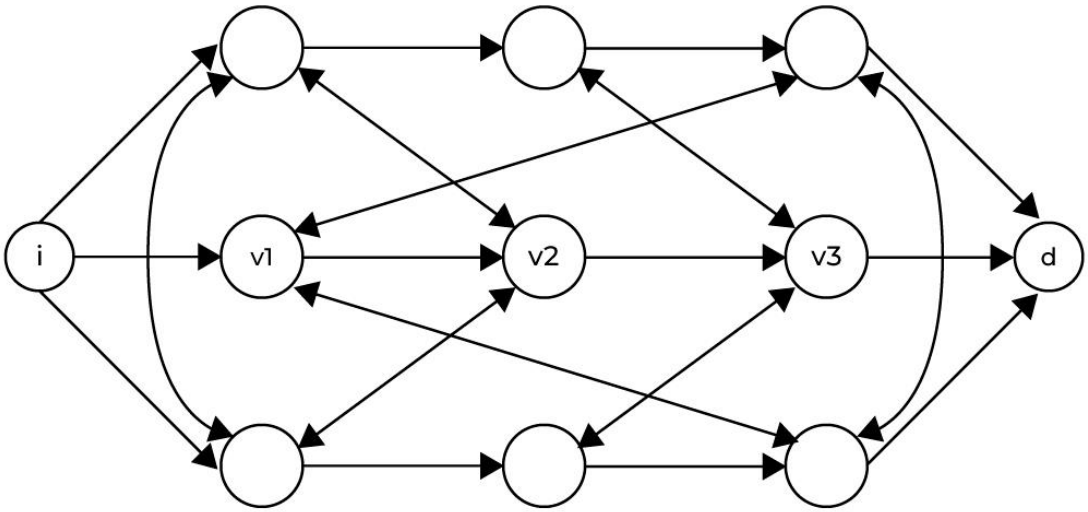
2 Figura 1.2. Gráfica de un árbol generado. Fuente: Varas & Alvarado

1.3.2 Shortest path problem

El problema del camino más corto ó *Shortest path problem* es uno de los más importante en el estudio de optimización y combinatoria, esto se debe a que es la raíz de otros problemas más complejos, en otras palabras, la formulación de una variedad de problemáticas se lo hace tomando en cuenta los componentes de estudio del camino más corto.

Como se menciona en él artículo (Aguilar, 2021) se trabaja con grafos dirigidos etiquetados o ponderados con factores de peso no negativos, es frecuente buscar el camino más corto entre dos vértices dados; es decir, el camino que nos permita llegar desde un vértice origen a un vértice destino recorriendo la menor distancia o con el menor costo.

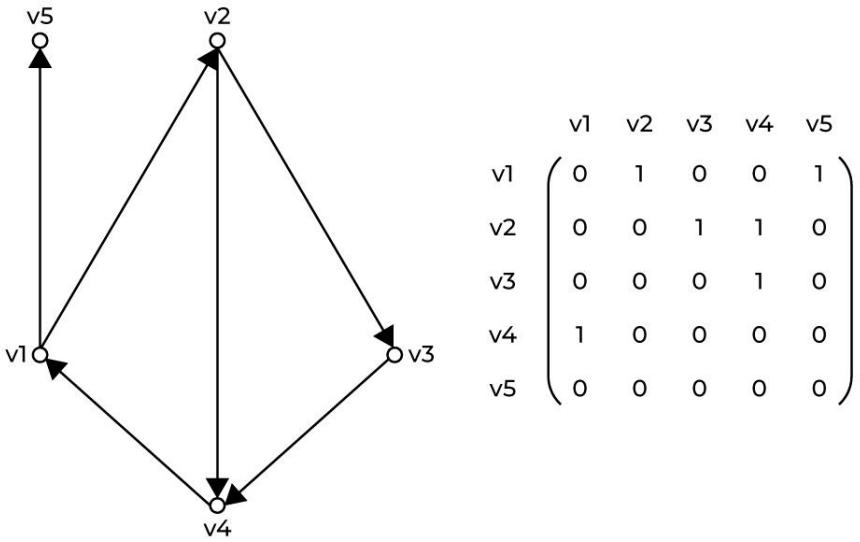
Este consiste en encontrar la ruta ó camino óptimo de un tipo de atributo que está asociado a cada una de las aristas que conectan a los nodos de una red, esta ruta escogida debe de tener un nodo de inicio y debe de llegar a un nodo destino pasando por distintos nodos de la red. El atributo que se busca minimizar normalmente representa la distancia o tiempo que se requiere en ir de un nodo a otro.



3 Figura 1.3. Gráfico de una red en búsqueda del camino más corto de un nodo i a un nodo d. Fuente: Varas & Alvarado

1.3.3 Matriz de adyacencia

Para obtener la matriz de adyacencia de este grafo, primero seleccionemos un ordenamiento de los vértices, digamos v_1, v_2, v_3, v_4, v_5. Luego, rotulemos las filas y columnas de una matriz con los vértices ordenados. El elemento en esta matriz es 1 si los vértices correspondientes a la fila y la columna son adyacentes y O en otro caso. (Lucca, 2015). En un grafo ponderado los valores de uno se reemplazan por los valores de los pesos que tienen las aristas de un grafo.



4 Figura 1.4. Grafo y su matríz de adyacencia. Fuente: Varas & Alvarado

1.3.4 Algoritmo de Dijkstra

Este es uno de los algoritmos de solución más destacados en el estudio de grafos, su nombre es en honor a Edsger Dijkstra quién lo dio a conocer hace ya medio siglo en 1959. Se trata de uno de los principales algoritmos que resuelve el problema de la ruta más corta en donde se encuentra el camino que minimice el costo de ir de un punto origen a un punto destino. En este existe una limitación como lo es el de los valores asociados a las aristas del grafo donde estos deben de ser no negativos.

El algoritmo de optimización que se usa en este método es el algoritmo Dijkstra. Este algoritmo funciona dado un grafo a cuyos arcos se han asociado una serie de pesos, se define el camino de coste mínimo de un vértice u a otro v, como el camino donde la suma de los pesos de los arcos que lo forman es la más baja entre las de todos los caminos posibles de u a v. (Lozano, 2001)

Dado un grafo cuyos arcos se han asociado una serie de pesos, se define el camino de coste mínimo de un vértice u a otro v, como el camino donde la suma de los pesos de los arcos que lo conforman es a más baja entre todos los caminos posibles,

El algoritmo de Dijkstra es eficiente, de complejidad $O(n^2)$, donde O es la complejidad y n el número de vértices. Este algoritmo sirve para encontrar el camino de coste mínimo desde un nodo origen a todos los demás nodos del grafo. (Lozano, 2001)

1.3.5 Pseudocódigo de Dijkstra

```
Entrada:
   Dígrafo G,          Grafo a estudiar.
   Pesos w:E(G) → ℝ+;   E(G)  es un conjunto de pares ordenados que
relaciona dos vértices.
   Vértice i ∈V(G) ;  donde i es el vértice de inicio

   Vértice d ∈V(G) ;  donde d es el vértice de destino objetivo

Salida:
   l(v) ∀v ∈ V − {i} ,  donde l(v) es el tiempo mínimo de llegar a v
desde i.
   p(v) ∀v ∈ V − {i} ,  donde p(v) es el predecesor de v.



1 Establecer
   l(i) = 0
   M = número muy grande.
   R = Ø

2 para todo v ∈ V-{i} hacer
   l(v) = M

3 Encontrar un vértice v donde v ∈ V(G) y v ∉ R; tal que l(v) = min
l(a) a ∈ V(G) y a ∉ R.

4 definir R = R ∪ {v}

5 Para todo a donde ∈ V(G) y a ∉ R; tal que (v, a) ∈ E(G) hacer;
   Si l(a) > l(v) + w((v, a)) entonces
    definir
    l(a)= l(v)+ w(a,v)
    p(a)=v


6 Si R≠ V(G) entonces volver al 3
```

5 Figura 1.5. Pseudocódigo del algoritmo de Dijkstra. Fuente: Varas & Alvarado

1.3.6 Algoritmo de Bellman Ford

El algoritmo de Bellman Ford genera los caminos mínimos desde un nodo de origen i de un grafo dirigido al resto de los nodos de este que se encuentran conectados, también soluciona el (SPP) desde un nodo de origen de manera más general que el algoritmo de Dijkstra, ya que este algoritmo permite valores negativos en los arcos.

Como indica *(Bellman, 1958)*, el algoritmo partirá de un vértice origen para escoger vértices de mínimo peso y restaurar sus distancias mediante el paso de relajación, Bellman-Ford simplemente relaja todas las aristas y lo hace $(|V| - 1)$ veces, siendo $|V|$ el número de vértices del grafo.

1.3.7 Pseudocódigo de Bellman Ford

```
Entrada:
   Dígrafo G,          Grafo a estudiar.
   Pesos w:E(G)→ℝ+;   E(G)  es un conjunto de pares ordenados que
relaciona dos vértices.
   Vértice i ∈V(G) ;  donde i es el vértice de inicio

   Vértice d ∈V(G) ;  donde d es el vértice de destino objetivo

Salida:
   l(v) ∀v∈V−{i} ,   donde l(v) es el tiempo mínimo de llegar a v
desde i.
   p(v) ∀v∈V−{i} ,   donde p(v) es el predecesor de v.



1 Establecer
   l(i) = 0
   M = número muy grande.
   n=|V(G)|

2 para todo v ∈ V-{i} hacer
   l(v) = M

3 Para todo i=1 a n-1 hacer
4  Para todo a donde ∈ V(G) y a ∉ R; tal que (v, a) ∈ E(G) hacer;
     Si l(a) > l(v) + w((v, a)) entonces
       definir
       l(a)= l(v)+ w(a,v)
       p(a)=v


6 Si R≠ V(G) entonces volver al 3
  Para todo a ∈V(G) y a ∉R; tal que (v,a) ∈E(G)
    Si l(a)>l(v)+w(v,a) entonces
     escribir tiene ciclo negativo
     parar
```

6 Figura 1.6. Pseudocódigo del algoritmo de Bellman Ford. Fuente: Varas & Alvarado

1.3.8 Diferencias Bellman Ford y Dijkstra

El algoritmo de Bellman-Ford nos da como resultado el camino más corto en un grafo dirigido ponderado y en el cuál él valor asociado a sus aristas puede ser menor a 0 ó negativos, mientras que el algoritmo de Dijkstra soluciona este idéntico problema en un tiempo menor, pero en sí el Algoritmo Bellman-Ford se emplea cuando hay aristas con peso negativo en comparación del algoritmo Dijkstra que las aristas de su red no debe

contener pesos negativos, a menos que el grafo sea dirigido y sin ciclos, caso contrario si existe pesos negativos y no el grafo no cumple con las condiciones mencionadas, este no generará una solución óptima.

1.3.9 Algoritmo de Floyd

El algoritmo de Floyd analiza el grafo de tal manera que encuentra el camino mínimo de forma dirigida y ponderada entre todos los pares de vértices de una única ejecución.

1.3.10 Pseudocódigo de Floyd

```
Entrada:
   Dígrafo G,          Grafo a estudiar.
   Pesos w:E(G) → ℝ⁺;   E(G)  es un conjunto de pares ordenados que relaciona
dos vértices.
   Vértice i ∈ V(G) ;  donde i es el vértice de inicio

1 Para todo k ∈ V − 1 hacer
2   Para todo i ∈ V − 1 hacer
3     Para todo j ∈ V − 1 hacer
4       Si w(i,k) + w(k,j) < w(i,j) entonces
5          w(i,j)= w(i,k) +w(k,j)
```

7 Figura 1.7. Pseudocódigo del algoritmo de Floyd. Fuente: Varas & Alvarado

1.3.11 Complejidad de Floyd y Dijkstra.

De acuerdo con (Arruabarrena, 2000) Algoritmo de Dijkstra de cálculo de las distancias mínimas de un nodo a todos los demás $O(v^2)$. Algoritmo de Floyd que calcula las distancias mínimas entre cada par de vértices $O(v^3)$.

1.3.12 Complejidad computacional del (TCSPP)

Los problemas de optimización que requieran moverse de una ciudad a otra o desde un nodo de inicio a uno o varios nodos de fin llegan a ser de gran complejidad él lo que se refiere a el tiempo de ejecución, uno de los objetivos cuando se diseña y prueba los algoritmos es el de encontrar cual es el método más óptimo que permita la menor utilización de recursos de memoria, debemos saber que el tiempo que se tarde realizando las iteraciones va a corresponder a la cantidad de nodos existentes y el tamaño de la entrada con los tiempos que se encuentran asignados.

Como menciona (Chávez, 2014) Es difícil realizar un análisis simple de un algoritmo que determine la cantidad exacta de tiempo que este requiere para ser ejecutado, porque depende en gran parte del algoritmo y de la computadora en que se ejecute. Además, conocer el tiempo exacto que tardará un programa de cómputo en dar resultados es una tarea difícil de determinar. En su lugar, es mejor calcular la cantidad de operaciones que se realizan de acuerdo con los datos de entrada del problema a tratar, lo que se conoce como cálculo de la función temporal.

Asimismo, con respecto al grado de complejidad de nuestro problema (Chávez, 2014) nos menciona que a través de ciertos modelos se han detectado problemas intratables, clasificados como NP *(nondeterministic polynomial time),* que se piensa son imposibles de resolver en un tiempo razonable cuando el número de variables que los componen es una cantidad extremadamente grande. Este grupo incluye problemas como el del agente viajero, el de la mochila, el transporte, la asignación de horarios para cursos universitarios o la asignación de maquinaria en talleres de manufactura.

CAPÍTULO 2

2. METODOLOGÍA

2.1 Técnicas de investigación

2.1.1 Levantamiento de información

Los datos utilizados en este proyecto fueron obtenidos por nosotros con el objetivo de crear una instancia de nuestra problemática a resolver. Nuestra técnica de investigación es experimental, debido a que ciertos datos y situaciones serán manipuladas por la complejidad para obtener un problema real en el cual se cumpla los criterios de nuestra problemática y, asimismo, lo difícil de adquirir datos de manera certera de un evento en específico. La obtención de información en campo se realizó en distintas intersecciones y calles del norte de la ciudad de guayaquil a inicio de horarios nocturnos.

Los datos de campo obtenidos fueron recopilados durante una semana, estos corresponden a los tiempos de espera para la autorización de circulación de ciertos semáforos en ciertos puntos de la ciudad y el tiempo promedio que tarda un auto en

circular desde uno de los puntos a otro en una franja horaria específica. Los datos fueron almacenados en un documento de Excel después de ser obtenidos.

ORIGEN	DESTINO	TIEMPO ESTIMADO 1	TIEMPO ESTIMADO 2	TIEMPO ESTIMADO 3	TIEMPO ESTIMADO 1 (ENTERO)	TIEMPO ESTIMADO 2 (ENTERO)	TIEMPO ESTIMADO 3 (ENTERO)	Promedio
i	v_1	0:00:31	0:00:38	0:00:29	0,52	0,63	0,48	0,54
v_1	v_2	0:03:10	0:03:59	0:03:45	3,17	3,98	3,75	3,63
v_2	v_6	0:02:03	0:02:15	0:02:24	2,05	2,25	2,40	2,23
v_6	d	0:05:45	0:05:56	0:05:02	5,75	5,93	5,03	5,57
i	v_3	0:03:55	0:03:42	0:03:44	3,92	3,70	3,73	3,78
v_3	d	0:12:53	0:12:23		12,88	12,38	0,00	8,42
v_3	v_2	0:05:58	0:05:12	0:05:24	5,97	5,20	5,40	5,52
v_3	v_4	0:02:25	0:02:43	0:02:32	2,42	2,72	2,53	2,56
v_4	v_2	0:04:32	0:04:34	0:04:13	4,53	4,57	4,22	4,44
v_4	d	0:17:34	0:17:13	0:17:13	17,57	17,22	17,22	17,33
v_4	v_5	0:03:53	0:04:10	0:04:36	3,88	4,17	4,60	4,22
v_5	v_2	0:00:54	0:01:24		0,90	1,40	0,00	0,77
v_2	d	0:11:12	0:12:56		11,20	12,93	0,00	8,04
v_5	v_7	0:01:34	0:02:02	0:01:40	1,57	2,03	1,67	1,76
v_7	d		0:07:32		0,00	7,53	0,00	2,51
v_1	d	0:13:47			13,78	0,00	0,00	4,59
v_2	v_5	0:00:43	0:00:32	0:00:38	0,72	0,53	0,63	0,63
v_7	v_6	0:00:27	0:00:22	0:00:25	0,45	0,37	0,42	0,41

Tabla 1 Hoja de cálculo de datos recopilados sobre el tiempo de ruta entre los nodos adyacentes. Fuente: Varas & Alvarado

2.1.2 Análisis de la información levantada.

Los datos obtenidos corresponden al tiempo estimado de circular de un punto de la ciudad a otro. Se consiguió realizar la toma de datos en la semana 28 del año 2022. Los sectores tomados en consideración son los derivados a tres vías principales del norte de la ciudad de Guayaquil, las cuales son: Av. Francisco de Orellana, Av. Vía Daule, Autopista Terminal Terrestre Pascuales.

El objetivo de la información levantada era conseguir los valores del tiempo recorrido de quienes serán nuestras aristas en la ejemplificación futura de la problemática y de tomar en cuenta ciertos datos de tiempos de esperas en los semáforos de 7 intersecciones para dar un mejor criterio o sentido a un escenario inexistente que planteamos nosotros en búsqueda de una situación real de nuestra problemática.

La información del tiempo de recorrido se la obtuvo mediante un cronómetro. Se recorrió las calles con un vehículo en un horario de 08:00:00 P.M A 10:30:00 P.M, horario el cuál se considera un nivel de tráfico no elevado.

2.2 Descripción del modelo

2.2.1 Nodos e intersecciones

2.2.1.1 Nodos origen y destino.

Para nuestro problema definimos los puntos de origen y de destino de nuestro grafo. En este caso, cómo problemática será obtener la ruta de tiempo más corta de ir desde Mabe, que se encuentra ubicado en la entrada principal de Pascuales, específicamente en la intersección de la Vía Daule y la Av. Principal Pascuales, hasta la entrada principal del Parque Samanes que se encuentra en la Av. Francisco de Orellana. Estos nodos se visualizarán en la **figura 2.2**.

Variable	**Nodo**
i	Origen
d	destino

Tabla 2 Nodos origen y destino. Fuente: Varas & Alvarado

2.2.1.2 Nodos del grafo.

Los otros nodos que formarán parte del grafo corresponden a distintas 7 intersecciones de la ciudad de Guayaquil. Estos se encuentran ubicados distintos partes del Norte de la urbe, donde las vías que intervienen en el grafo son las que se muestra en la **tabla 3**.

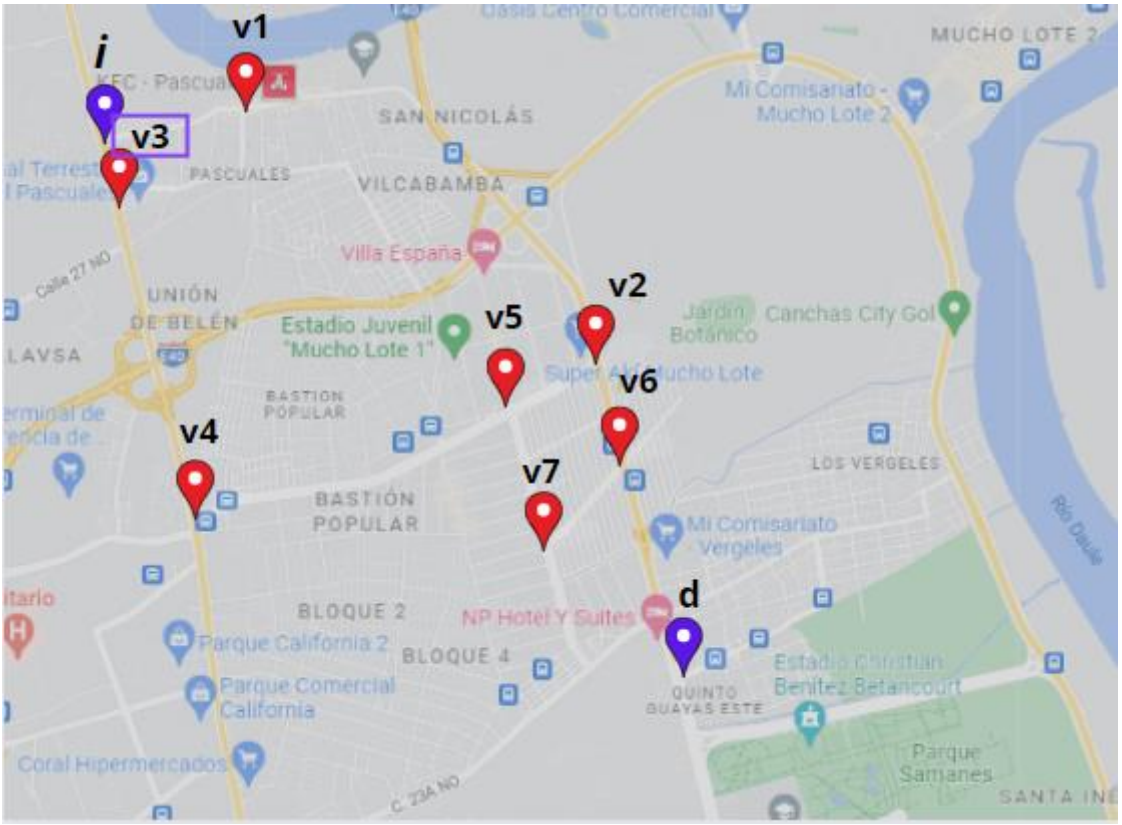
CALLES QUE INTERVIENEN EN EL GRAFO	**REFERENCIA**
Av. Camilo Ponce Enrique	Vía Daule. Desde KM 16 Hasta el KM 12 ½. Sentido Norte-Centro.
Av. Manuel Ignacio Gómez Lince	Desde su inicio del KM 12 ½ Vía Daule hasta donde conecta con Av. Francisco de Orellana.
Av. Francisco de Orellana	Desde entrada de Pascuales hasta el Parque Samanes.
Av. Principal Pascuales	Desde Mabe hasta salida Av. Francisco de Orellana
Av. 33 Mucho Lote	Redonde de Almacenes Tía de mucho Lote hasta el fin del parque lineal de Mucho Lote

Tabla 3 Calles y avenidas que son parte de los nodos del grafo. Fuente: Varas & Alvarado

Las intersecciones son las que se detallan en la **tabla 4**. Así de esta manera se definieron los nodos que formarán parte del Grafo.

INTERSECCIÓN	*VARIABLES*	REFERENCIA- DIRECCIÓN
$Intersección\ 1$	v_1	Calle principal de Pascuales - Av. Salitre.
$Intersección\ 2$	v_2	Av. Francisco de Orellana - Av. Manuel Ignacio Gómez
$Intersección\ 3$	v_3	Av. Camilo Ponce Enrique - Terminal Municipal Pascuales.
$Intersección\ 4$	v_4	Dr. Marcel J Laniado de Wing - Vía Daule-Av. Camilo Ponce Enrique
$Intersección\ 5$	v_5	Av. Manuel Ignacio Gómez Lince - Av. 33 NO-P
$Intersección\ 6$	v_6	Av. Francisco de Orellana - C.24B N-O
$Intersección\ 7$	v_7	Av. 33 N-O - C24 N-O

Tabla 4 Nodos e intersecciones definidos del grafo. Fuente: Varas & Alvarado



8 Figura 2.2. Visualización gráfica de las intersecciones del grafo. Fuente: Varas & Alvarado

2.2.2 Aristas.

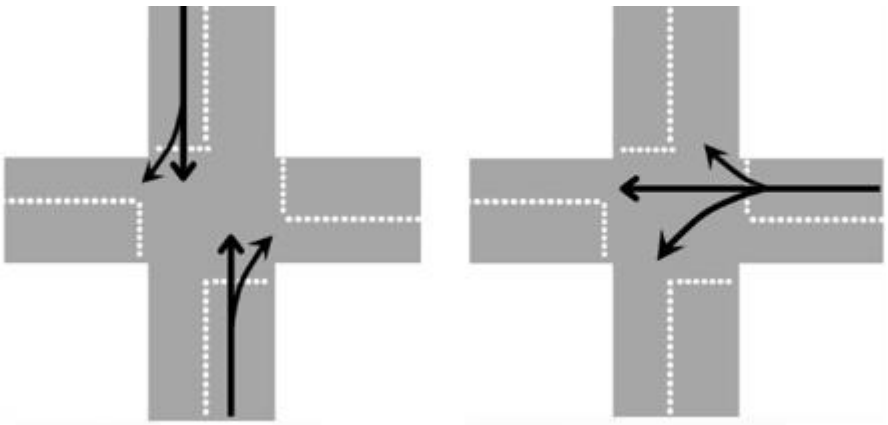
Para el problema se tomaron en cuenta 18 recorridos principales que conectan a estas intersecciones alrededor de la ciudad. Estos corresponderán a las aristas de nuestro grafo y se detallan en la **tabla 5.**

Arista	ORIGEN	DESTINO
a_1	i	v_1
a_2	v_1	v_2
a_3	v_2	v_6
a_4	v_6	d
a_5	v_1	v_3
a_6	v_3	d
a_7	v_3	v_2
a_8	v_3	v_4
a_9	v_4	v_2
a_{10}	v_4	d
a_{11}	v_4	v_5
a_{12}	v_5	v_2
a_{13}	v_2	d
a_{14}	v_5	v_7
a_{15}	v_7	d
a_{16}	v_1	d
a_{17}	v_2	v_5
a_{18}	v_7	v_6

Tabla 5 Aristas definidas del Grafo. Fuente: Varas & Alvarado

2.2.3 Control de luces de las intersecciones

Para nuestra toma de datos, definimos el control de luces de cada una de las intersecciones, como las distintas rutas que un conductor puede tomar a partir de que el semáforo de esa intersección se lo permita y que proviene desde un punto dirigiéndose a otro. Este término se lo considera también como las fases de un semáforo. La **figura 2.3** se trata de una representación de una intersección que tiene dos fases.

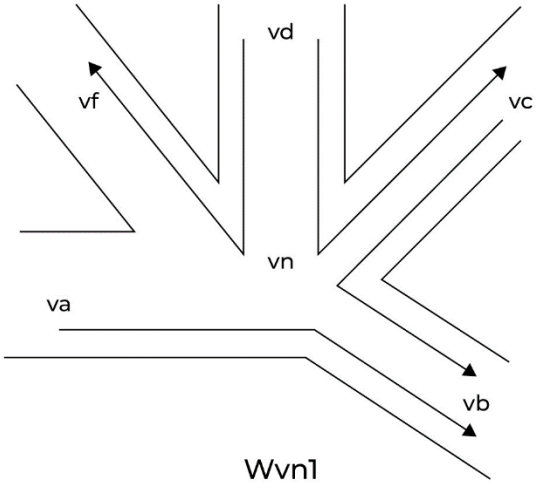


8 Figura 2.3. Representación gráfica de las fases de intersección. Fuente: Varas & Alvarado

2.2.4 Ventanas de tiempo

Dado conjunto n de nodos que intervienen en el grafo y el conjunto i de ventanas de tiempo. Se define el conjunto $P_{v_n,i}$ de rutas permitidas en la $i-ésima$ ventana de tiempo $ws_{v_{n,i}}$ para la intersección o nodo v_n, donde estas rutas están representadas por tripletas de la forma $< v_s, v_t, v_v >$ y que indica la ruta que proviene de la intersección v_m pasando por la intersección v_n y dirigiéndose v_p.

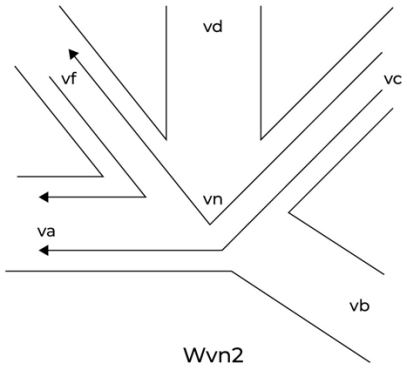
En la **figura 2.4** se observa que en la intersección v_n para la ventana de tiempo $ws_{v_{n,1}}$ se pueden tomar las siguientes rutas:



9 Figura 2.4. Representación gráfica de las rutas permitidas de una de las fases de una intersección. Fuente: Varas & Alvarado

$$P_{v_n,1} = \left\{\langle v_a, v_n, v_b\rangle, \langle v_d, v_n, v_f\rangle, \langle v_d, v_n, v_c\rangle, \langle v_c, v_n, v_b\rangle\right\}$$

Mientras que para la ventana de tiempo $w_{v_{n,2}}$ se puede realizar las siguientes rutas como se muestra en la **figura 2.5**.



10 Figura 2.5. Representación gráfica de las rutas permitidas de una de las fases de una intersección. Fuente: Varas & Alvarado

$$P_{v_n,2} = \left\{ \left\langle v_f, v_n, v_a \right\rangle, \langle v_c, v_n, v_f \rangle, \langle v_c, v_n, v_a \rangle \right\}$$

Además, para cada nodo v_n existe una variable wsi_{v_n} que representa el inicio de la primera ventana de tiempo desde el tiempo $t = 0$. Es, aquí uno de los temas complejos en la obtención datos, debido a que para conseguir este valor wsi_{v_n} para cada una de las intersecciones, se debe de estar en cada una de estas y empezar a obtener el cálculo de esta variable en un tiempo t específico hasta obtener el inicio de la primera ventana. Es decir, si el tiempo se lo obtiene mediante un cronometro, debería de existir uno por cada intersección y el inicio de la toma del tiempo debe de ser en el mismo instante.

2.2.5 Duración de la ventana de tiempo

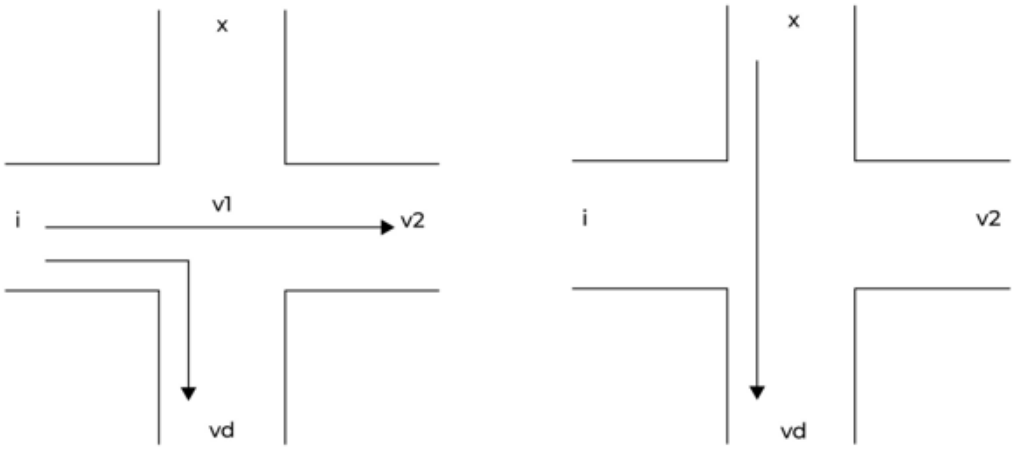
A cada una de nuestras ventanas de tiempo, se le asignó una variable representada por un valor de tiempo $d_{v_n,k}$ que denota la duración de la ventana de tiempo k para el vector v_n. La adquisición de los datos del tiempo de duración de cada una de las fases se realizó en cada una de las intersecciones. Para esto se consideraron solo las fases que dan circulación a las rutas que están relacionadas con nuestros grafos, es decir, una intersección en la cual tenga 3 fases y una de ellas no está relacionada con las rutas de nuestro grafo, no se tomó en cuenta, por lo que consideramos que esta intersección solo tendrá dos fases.

2.2.6 Tripletas del problema

Para las tripletas definidas de nuestro grafo se plantero un escenario que no es parte de la realidad, esto debido a la complejidad de encontrar un escenario en el cual se cumpla las condiciones de que en cada una de las intersecciones se tome en cuenta en todas sus ventanas de tiempo de cada una de ellas, se considera un nodo predecesor y un

nodo destino que también forme parte del problema.

Cómo lo muestra en la **figura 2.6** donde este representa la intersección 1 de la Calle principal de Pascuales y Av. Salitre v_1 de nuestro grafo. Para esta existen dos ventanas de tiempos en las cuales se pueden realizar las siguientes rutas.



11 Figura 2.6. Representación gráfica de las fases de una intersección de nuestra problemática. Fuente: Varas & Alvarado

Se puede observar que existe un destino u origen que no está definido en nuestro grafo cómo lo es el nodo x. Este problema es el más común a la hora de buscar este tipo de problemática. Es por esta razón que se creó un nuevo escenario.

2.3 Recopilación del problema

2.3.1 Tripletas del problema.

Para nuestro problema planteamos las siguientes tripletas para cada una de las intersecciones.

$\boldsymbol{nodo\ k}$	$\boldsymbol{ws_{v_{k,1}}}$	$\boldsymbol{ws_{v_{k,2}}}$
v_1	$\langle i, v_1, d\rangle$	$\langle i, v_1, v_2\rangle$
v_2	$\langle v_1, v_2, v_6\rangle, \langle v_4, v_2, v_6\rangle \langle v_4, v_2, v_5\rangle$	$\langle v_1, v_2, d\rangle$
v_3	$\langle i, v_3, v_4\rangle \langle i, v_3, v_2\rangle$	$\langle i, v_3, d\rangle$
v_4	$\langle v_3, v_4, d\rangle \langle v_3, v_4, v_2\rangle$	$\langle v_3, v_4, v_5\rangle$
v_5	$\langle v_3, v_5, v_2\rangle$	$\langle v_2, v_4, v_7\rangle$

v_6	$\langle v_2, v_6, d\rangle$	$\langle v_7, v_6, d\rangle$
v_7	$\langle v_5, v_7, d\rangle$	$\langle v_5, v_7, v_6\rangle$

Tabla 6 Tripletas definidas del problema. Fuente: Varas & Alvarado

2.3.2 Pesos de las aristas.

Los pesos de las aristas corresponderán al tiempo promedio recorrido de un nodo v_i a un nodo v_j. Los tiempos para promediar pertenecen a los datos que fueron obtenidos por nosotros y que serán los pesos $w(v_i, v_j)$ de las aristas del grafo G. En la mayoría de las aristas se tomaron en cuenta tres valores de tiempo a promediar que se observan en la **tabla 7**, sin embargo, existen arista que solo tiene uno ó dos valores así que de igual manera se tomará el promedio de sus valores obtenidos.

ORIGEN	**DESTINO**	**TIEMPO ESTIMADO 1 (hh:mm:ss)**	**TIEMPO ESTIMADO 2 (hh:mm:ss)**	**TIEMPO ESTIMADO 3 (hh:mm:ss)**
i	v_1	0:00:31	0:00:38	0:00:29
v_1	v_2	0:04:10	0:06:23	0:04:38
v_2	v_6	0:02:03	0:02:43	0:02:17
v_6	d	0:08:35	0:08:42	0:06:14
i	v_3	0:04:34	0:06:06	0:04:56
v_3	d	0:12:53	0:17:09	0:00:00
v_3	v_2	0:07:12	0:07:36	0:06:36
v_3	v_4	0:03:25	0:04:29	0:03:25
v_4	v_2	0:04:32	0:06:20	0:05:06
v_4	d	0:17:34	0:19:57	0:19:36
v_4	v_5	0:04:53	0:06:34	0:05:29
v_5	v_2	0:00:54	0:03:10	0:00:00
v_2	d	0:11:12	0:17:42	0:00:00
v_5	v_7	0:01:34	0:09:37	0:02:33
v_7	d	0:00:00	0:09:18	0:00:00
v_1	d	0:13:47	0:00:00	0:00:00
v_2	v_5	0:00:43	0:00:32	0:01:31
v_7	v_6	0:00:27	0:00:22	0:00:25

Tabla 7 Valores obtenidos de ir de un punto a un destino. Fuente: Varas & Alvarado

$$w(i,j) = \frac{\sum_i^n tiempo\ en\ valores\ enteros}{Número\ de\ datos\ obtenidos\ sobre\ el\ recorrido\ \ i\ a\ j}$$

Los valores del tiempo relacionado a cada una de las aristas son los que se muestran en la **tabla 8.**

ORIGEN	DESTINO	Arista	$w(i,j)(minutos)$
i	v_1	a_1	0,54
v_1	v_2	a_2	5,06
v_2	v_6	a_3	2,35
v_6	d	a_4	7,84
i	v_3	a_5	5,20
v_3	d	a_6	15,02
v_3	v_2	a_7	7,13
v_3	v_4	a_8	3,77
v_4	v_2	a_9	5,32
v_4	i	a_{10}	19,04
v_4	v_5	a_{11}	5,64
v_5	v_2	a_{12}	2,03
v_2	i	a_{13}	14,45
v_5	v_7	a_{14}	4,58
v_7	i	a_{15}	9,30
v_1	i	a_{16}	13,78
v_2	v_5	a_{17}	0,92
v_7	v_6	a_{18}	0,41

Tabla 8 Valores de los pesos del grafo obtenidos mediante el promedio de datos. Fuente: Varas & Alvarado

2.3.3 Duración de ventanas de tiempo

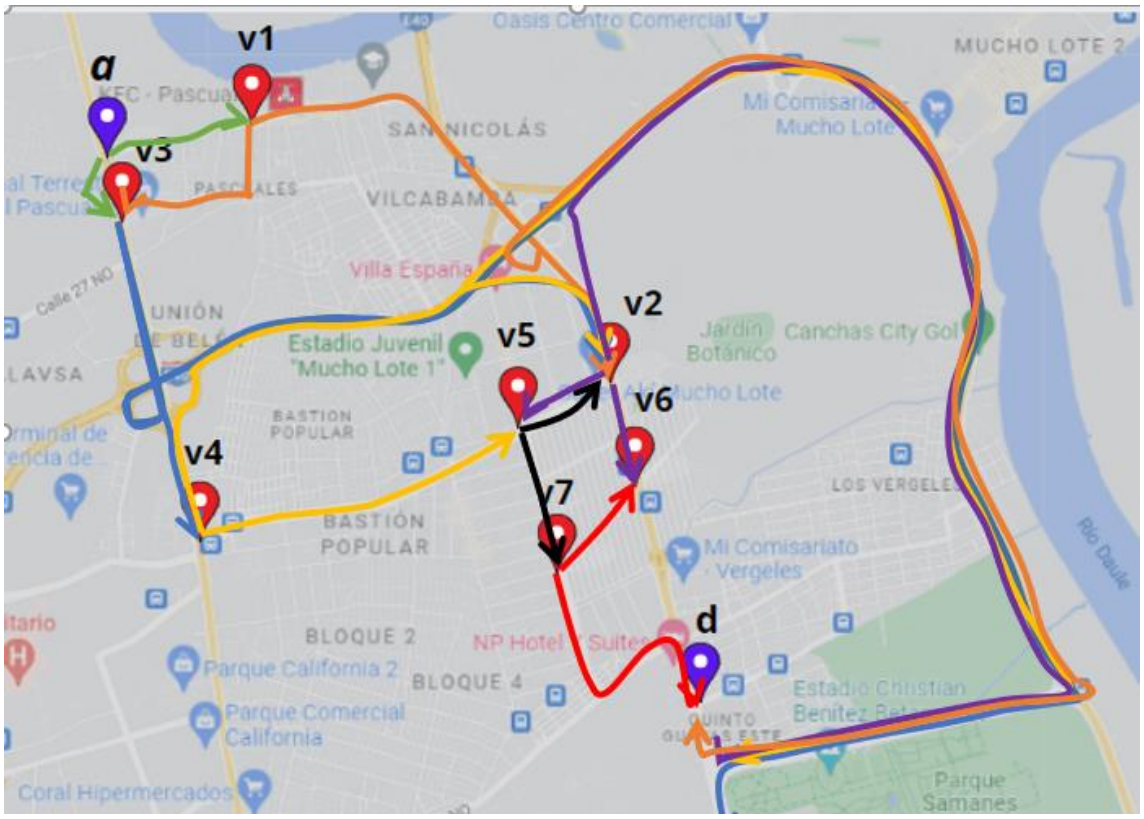
$nodo\ k$	$d_{v_{n,1}}$ (hh:mm:ss)	$d_{v_{n,2}}$ (hh:mm:ss)
v_1	0:00:34	0:00:45
v_2	0:00:43	0:00:40
v_3	0:00:38	0:00:40
v_4	0:00:45	0:00:35
v_5	0:00:55	0:00:45
v_6	0:00:43	0:00:40
v_7	0:00:35	0:00:35

Tabla 9 Data de la duración de las ventanas de tiempo. Fuente: Varas & Alvarado

2.3.4 Inicio de ventana de tiempo

Como se mencionó anteriormente, este es uno de los valores más complejos de obtener debido a la complejidad de obtener la toma de estos valores en el mismo instante de tiempo y en distintos lugares. Debido a esto, este será un valor aleatorio entre 0 y 0,92, ya que se conoce que 0:00:55 es el valor máximo duración de las ventanas de tiempo.

2.3.5 Grafo de nuestra problemática



12 Figura 2.7. Representación gráfica de las rutas obtenidas de la problemática. Fuente: Varas & Alvarado

2.4 Fases del proyecto

FASE 1

Definir correctamente el tipo de problemática a resolver, las variables y que condiciones debe cumplirse para hacer uso de la aplicación de la solución.

FASE 2

Plantear un escenario sobre nuestra problemática. Definir las variables que intervienen hasta definir un grafo con las respectivas ventanas de tiempo y su distancia, tomando en cuenta las consideraciones del problema y las limitaciones de encontrar un problema en un escenario real.
FASE 3
Programar un algoritmo que resuelva nuestro problema planteado y que este considere las diferentes variables que diferencian los diferentes escenarios de la problemática, por ejemplo: ventanas de tiempo, distancias de las ventanas de tiempo, entre otros. Dónde estos nos brinden la solución óptima del camino más corto.
FASE 4
Elaborar un modelo de programación lineal el cuál resuelva nuestro problema definido. Se procura tener un modelo que posea una función objetivo que nos brinde los valores óptimos de las variables de decisión específicas que determine la ruta óptima de ir de un punto de origen a un punto destino, así mismo, restricciones que permiten modelar los escenarios y limitaciones de nuestra red.

Tabla 10 Fases del proyecto. Fuente: Varas & Alvarado

2.5 Uso de software

Para la realización y ejecución del algoritmo de solución se usaron los softwares descritos a continuación, estos nos ayudan a ingresar la data que será utilizada, visualizar gráficamente nuestra red y programar nuestro algoritmo.

Microsoft Excel: En este proyecto se utilizaron las hojas de cálculo de Excel para el uso y manejo de la información recopilada. Además de que se utilizó para leer los datos por parte del software de programación.

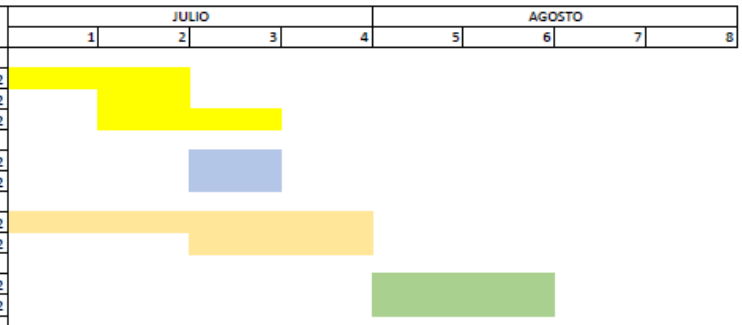
Phyton: Este lenguaje de programación permite programar nuestros algoritmos de solución del problema. Para ellos se utilizó la librería Digraph que nos ayuda a crear un grafo dirigido, numpy para la manipulación y tratamiento de datos.

Gams: Se utilizó este lenguaje para modelización del modelo de programación no lineal de nuestra problemática. El programa GAMS consiste en un lenguaje de modelización (módulo base) y un conjunto de solvers (algoritmos de resolución de problemas)

integrados a elección del usuario (dependiendo del tipo de problema que se pretenda resolver). En la Tabla 1 se muestran algunos de los solvers más utilizados. Además, puede usar GAMS en diferentes plataformas y utilizar muchas herramientas de conectividad y productividad. (Blanco, 2019).

1.6 Cronograma de trabajo

Actividades	FECHA INICIIO	HORAS ESTIMADAS	FECHA FIN
Recopilación de Información			
Datos del tiempo transcurrido entre nodos adyacentes.	10/7/2022	3	17/7/2022
Visita de las intersecciones	13/7/2022	2	13/7/2022
Datos del tiempo de duración de las ventanas de tiempo	15/7/2022	3	18/7/2022
Planteamiento del problema			
Ingreso y manipulación de data	18/7/2022	2	18/7/2022
Definición de variables, parámetros.	17/7/2022	3	17/7/2022
Implementación del modelo			
Programar el algoritmo de Python	1/7/2022	2	30/7/2022
Diseñar el modelo matemático en GAMS	30/7/2022	10	6/8/2022
Entregables			
Pseudocódigos y algoritmos	6/8/2022		10/8/2022
Código de programación	6/8/2022		10/8/2022
Modelo de programación			
Informe Final			



13 Figura 2.8. Cronograma del trabajo del proyecto. Fuente: Varas & Alvarado

CAPÍTULO 3

3. Resultado y análisis

3.1 Resultado de programación de las heurísticas

Durante el proceso de programación de las Heurística en el lenguaje Python, se creó un código para el algoritmo de Dijkstra, Bellman Ford y un modificado para cada uno de estos en el que se consideró la variante de semaforización. Estos códigos al ser ejecutado nos mostraron la respuesta al problema del camino más corto de nuestro grafo ya definido en el capítulo anterior.

3.1.1 Resultado de la programación de algoritmo de Dijkstra y Bellman Ford.

Se procedió a programar estos algoritmos porque son los que actualmente se consideran para resolver nuestra problemática y, además, por la razón de querer comparar el

resultado de la solución con el de los algoritmos modificados de nuestra variante, esto con el objetivo de tener un mejor análisis.

Cómo se sabe, estos algoritmos no toman en cuenta restricciones en los nodos, por lo que sé aclara que se omitió estas restricciones al momento de programar estos algoritmos, pero se tomó en cuenta de igual manera, los nodos, pesos y aristas definidos anteriormente.

Los resultados son los que se pueden observar en la **tabla 11**

NODOS	***DISTANCIA MAS CORTA (DIJKSTRA) (Minutos)***	**PREDECESOR**	***DISTANCIA MAS CORTA (BELLMAN) (Minutos)***	**PREDECESOR**
i	0	-	0	-
v_1	0,544	i	0,544	i
v_2	5,6	v_1	5,6	v_1
v_3	5,2	i	5,2	i
v_4	8,972	v_3	8,972	v_3
v_5	6,5277	v_2	6,5277	v_2
v_6	7,95555	v_2	7,95555	v_2
v_7	11,1047	v_5	11,1047	v_5
d	14,3277	v_1	14,3277	v_1

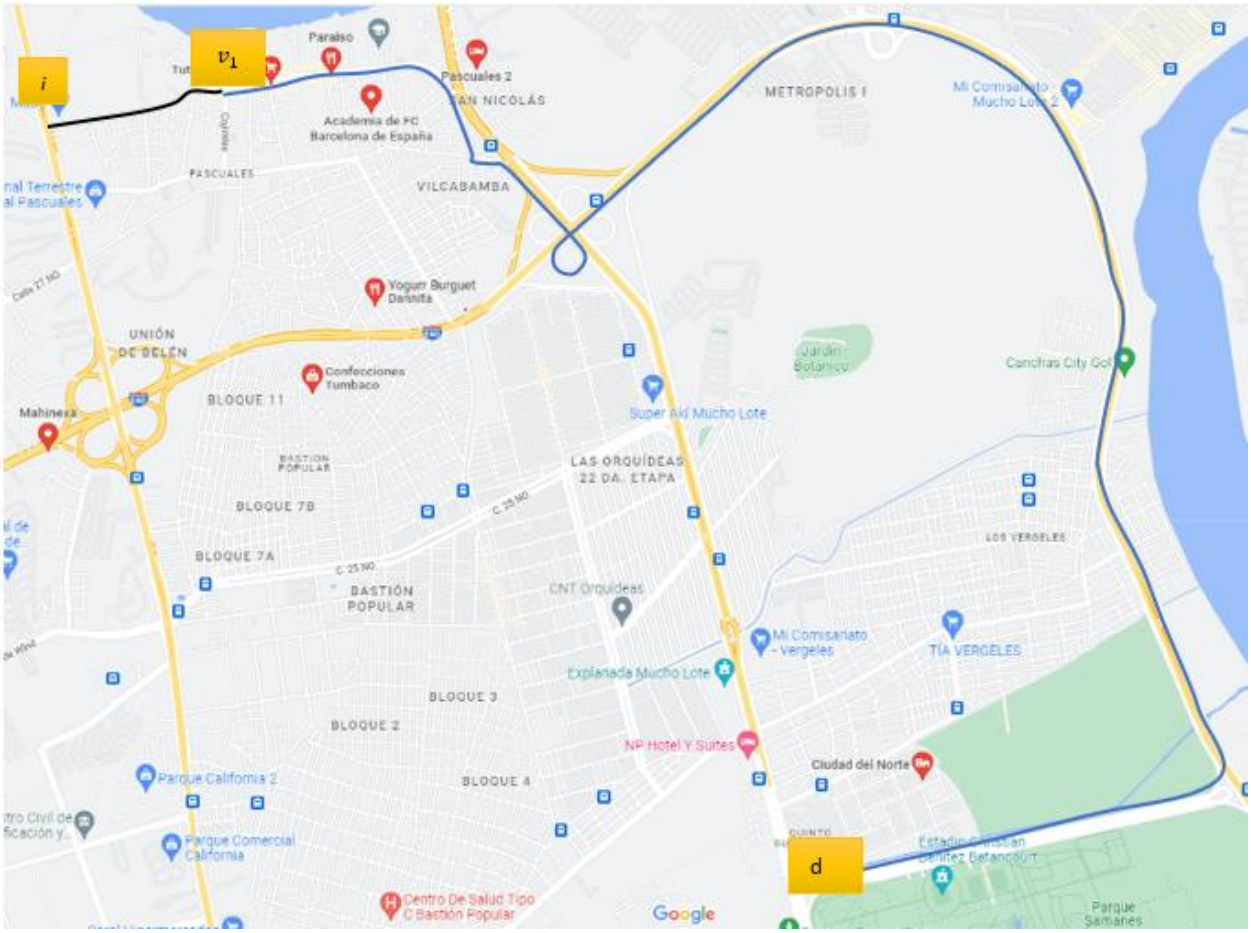
Tabla 11 Resultados obtenidos en la ejecución de algoritmo de Dijkstra y Bellman

Cómo ambos algoritmos resuelven el problema del camino más corto, se tiene como resultado los mismos valores de **tiempo mínimo recorrido desde el nodo origen a cada uno de los nodos** y el de **los predecesores que forman parte de estos caminos**.

Con esto tenemos que el camino más corto y el tiempo de este en ir del punto i al d en ambas ejecuciones es el siguiente.

$$i \rightarrow \quad v_1 \quad \rightarrow \quad d$$

$$Origen(MABE) \rightarrow Calle\ principal\ de\ Pascuales, Av. Salitre. \rightarrow\ Destino(Parque\ Samanes)$$



14 Figura 3.1. Ilustración del camino más corto del problema planteado. Fuente: Varas & Alvarado

$$\boldsymbol{Tiempo\ de\ recorrido} = 14{,}3277$$

3.1.2 Resultado de la programación de algoritmo de Dijkstra y Bellman Ford modificados.

Se procedió a programar nuestra heurística de solución para nuestro problema con variante establecida que es la de restricciones en los nodos. Para este caso, se tomó todos los datos mencionados en el capítulo 2.

Se debe de mencionar que al ejecutar nuestros códigos para los algoritmos modificados se obtuvieron como resultado **los valores de tiempo mínimo recorrido desde el nodo origen a cada uno de los nodos** y **los predecesores** de cada uno de ellos. Estos resultados son los siguientes.

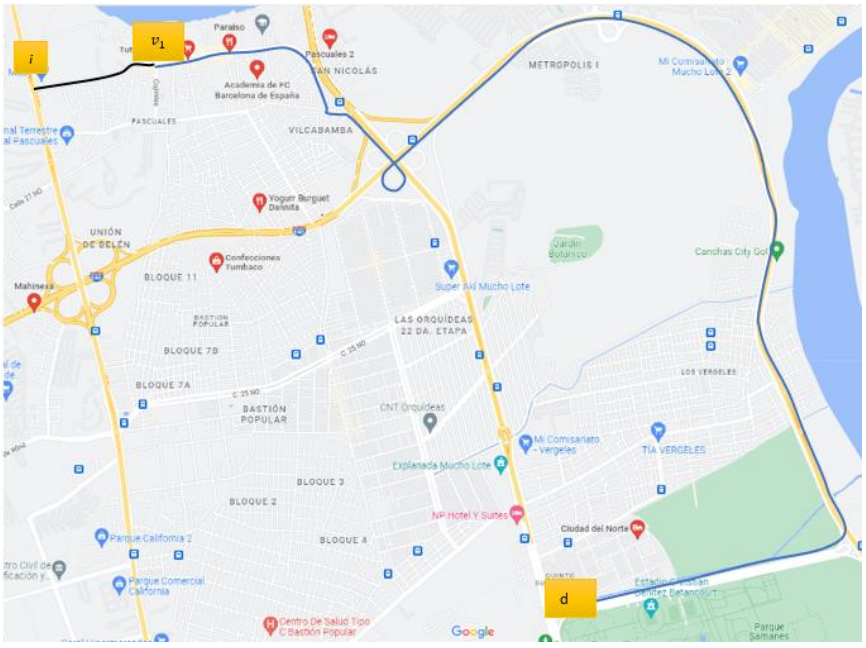
NODOS	DISTANCIA MAS CORTA (DIJKSTRA MODIFICADO) (Minutos)	PREDECESOR	DISTANCIA MAS CORTA (BELLMAN MODIFICADO) (Minutos)	PREDECESOR
i	0	-	0	-
v_1	0,544	i	0,544	i
v_2	5,605499	v_1	5,605499	v_1
v_3	5,2	i	5,2	i
v_4	9,182	v_3	9,182	v_3
v_5	6,71222	v_2	6,71222	v_2
v_6	8,14	v_2	8,14	v_2
v_7	11,289222	v_5	11,289222	v_5
d	14,3277	v_1	14,3277	v_1

Al igual que en la situación actual, si se realiza una modificación de estos algoritmos para resolver nuestro problema planteado de restricciones en los nodos, se tendrá las mismas soluciones ya antes mencionadas tanto para el algoritmo Dijkstra modificado como el de Bellman Ford modificado al ejecutar el algoritmo.

Definiendo esto, la respuesta para el problema mencionado de ir a Mabe hasta Samanes tomando en cuenta las restricciones en los nodos, es la siguiente.

$$i \rightarrow v_1 \rightarrow d$$

$$Origen(MABE) \rightarrow Calle\ principal\ de\ Pascuales, Av. Salitre. \rightarrow Destino(Parque\ Samanes)$$



15 Figura 3.2. Ilustración del camino más corto del problema planteado. Fuente: Varas & Alvarado

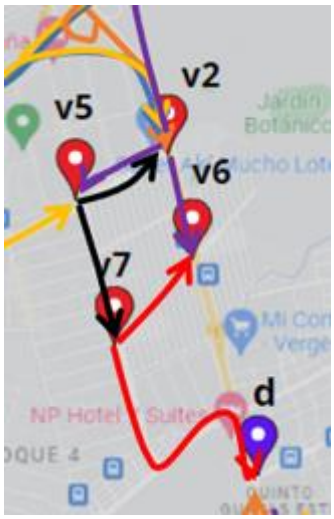
$$\boldsymbol{Tiempo\ de\ recorrido} = 14{,}3277\ minutos$$

3.2 Análisis de los resultados

3.2.1 Análisis de resultados obtenidos.

A pesar de que ciertos datos fueron modificados como se menciona en el anterior capítulo, con el objetivo de replicar la situación real y definir un problema de este tipo, se pudo obtener una respuesta considerable comparado con la realidad. Al momento de hacer la investigación de campo para la obtención de datos para el capítulo 2, se pudo conocer con respecto al camino más corto de este problema establecido que este podría ser el camino óptimo debido al tiempo que sé estimó en el transcurso de ir en el vehículo del punto de origen al punto destino obteniendo los datos. Esto se debe a que en el camino de solución no se origina mucho tráfico comparado con el de las otras avenidas que forman parte de las aristas del grafo, como es el de la Av. Francisco de Orellana y Vía Daule, avenidas cuales se encuentran gran cantidad de los nodos de nuestro grafo.

Asimismo, en cuanto a los nodos también se podían predecir los potenciales predecesores debido a la cercanía con los demás, cómo se puede observar en las siguientes **figuras 3.3 y 3.4**.



16 Figura 3.3. Ilustración de aristas y nodos del grafo. Fuente: Varas & Alvarado

17 Figura 3.4. Ilustración de aristas y nodos del grafo. Fuente: Varas & Alvarado

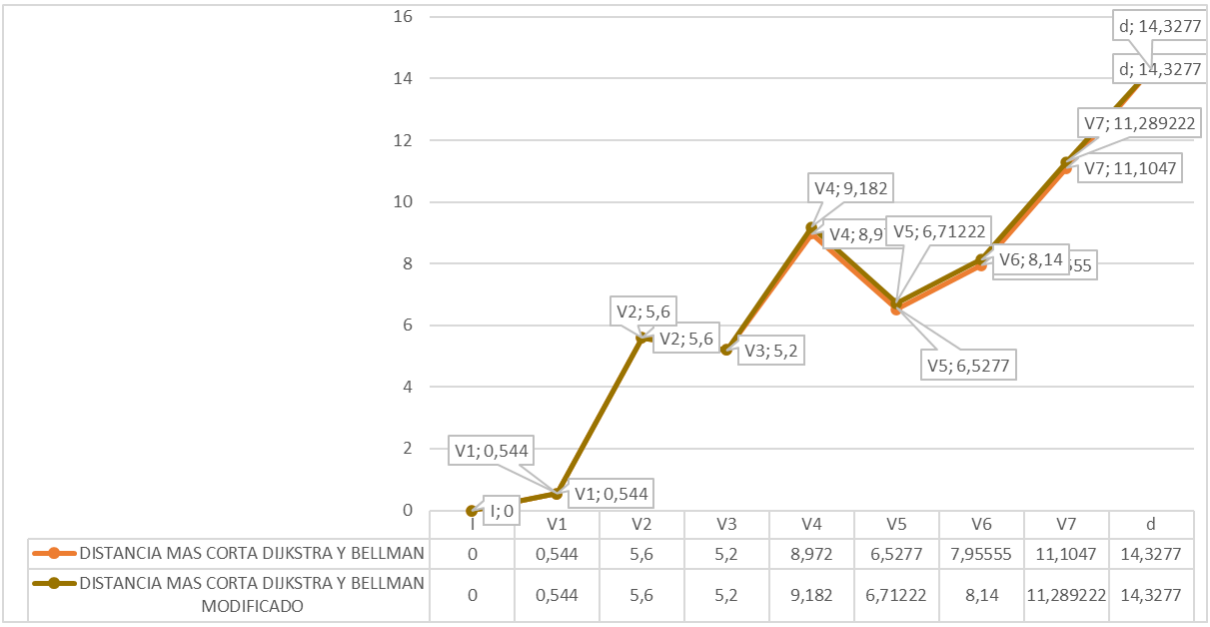
En la **figura 3.3** se puede observar una gran cercanía entre los nodos v_2, v_5, v_6, v_7 por lo que los pesos de aristas o el tiempo de dirigirse de un nodo de estos a otro es muy corto cómo se puede observar en la **Tabla 7**, el tiempo de ir v_7 a v_6 no supera los 30 segundos. El mismo criterio es para los nodos de la **figura 3.4**. Este también podría ser una de las situaciones por la cual se genera tráfico en estás vías, ya que cada nodo representa una intersección con semaforización y la cercanía de estas muestra la poca fluides que se da a cabo en estos recorridos o caminos.

3.3 Comparación de resultados actuales con propuesta.

3.3.1 Camino más corto

Como se mencionó anteriormente, una de las decisiones actuales al momento de escoger un algoritmo de solución para el problema del camino más corto, son los algoritmos de Dijkstra y el de Bellman Ford, lo que se busca es dar una opción más a la hora de tomar esta decisión y esto son tener como una posible elección los algoritmos modificados que fueron planteados y que fueron implementados en un código de programación.

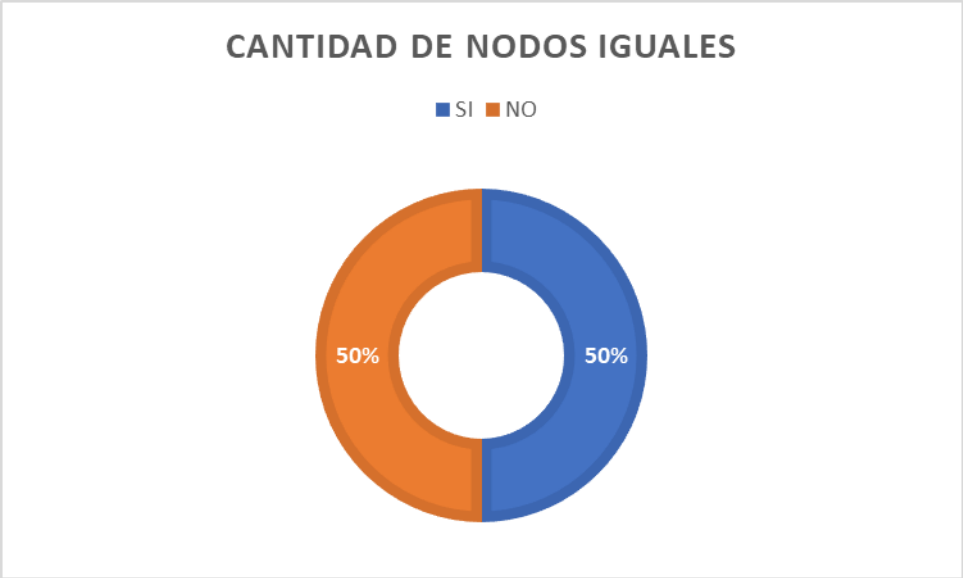
Se procedió hacer una comparación de estos resultados comenzando por la variación de los valores de distancia mínimas. En la **figura 3.5** se puede observar la variación de las distancias mínimas de ir del punto de origen i a cada uno de los nodos por parte de los algoritmos de Dijkstra. Se distingue que se tiene se tiene una mayor variación entre los nodos que tienen cómo solución óptima un camino con un mayor número de predecesores.



	I	V1	V2	V3	V4	V5	V6	V7	d
DISTANCIA MAS CORTA DIJKSTRA Y BELLMAN	0	0,544	5,6	5,2	8,972	6,5277	7,95555	11,1047	14,3277
DISTANCIA MAS CORTA DIJKSTRA Y BELLMAN MODIFICADO	0	0,544	5,6	5,2	9,182	6,71222	8,14	11,289222	14,3277

18 Figura 3.5. Gráfica de variación de tiempos óptimos para situación actual y propuesta. Fuente: Varas & Alvarado

A pesar de que existe una variación de las distancias, existen nodos que tiene la misma solución óptima de distancia más corta tanto para la situación actual y en el de nuestra propuesta. Como se refleja en la **figura 3.6,** existe un 50% del total que mantienen sus distancias óptimas iguales, entre ellas la solución óptima que buscamos la cuál es la de camino óptimo de ir de i a d, esto es debido a que para llegar al nodo de destino se tiene solo un único predecesor que posee restricciones de semaforización y este es e el vector v_1. Esto se puede entender cómo un vehículo que partió del origen y solo tuvo que pasar por una intersección la cuál al llegar estaba en verde y es por esto por lo que no existe un cambio en el tiempo óptimo.



19 Figura 3.6. Grafica de porcentaje de nodos iguales. Fuente: Varas & Alvarado

3.3.2 Predecesores

Con respecto a los predecesores de cada uno de los nodos, se obtuvieron los mismos resultados en todos los algoritmos de solución programados, esto es debido a lo antes ya mencionado con respecto a la cercanía de los nodos y la notable diferencia entre los pesos de las aristas del grafo.

También hay que agregar otras situaciones a considerar del grafo planteado, una de ellas es que la mayoría de los nodos solo tienen cómo posibles predecesores a uno o dos candidatos ó en otras palabras, gran parte de los nodos del grafo solo es adyacente a uno o dos de este cómo lo es el vector de destino.

NODOS	**PREDECESORES DIJKSTRA Y BELLMAN**	**PREDECESORES DIJKSTRA Y BELLMAN MODIFICADOS**
i	-	-
v_1	*i*	*i*
v_2	v_1	v_1
v_3	i	i
v_4	v_3	v_3
v_5	v_2	v_2
v_6	v_2	v_2
v_7	v_4	v_4
d	v_1	v_1

Tabla 12 Solución de los predecesores de la situación actual y propuesta. Fuente: Varas & Alvarado

3.4 Pseudocódigo de Dijkstra modificado

```
Entrada:
   Dígrafo G,          Grafo a estudiar.
   Pesos w:E(G) → ℝ+;   E(G)  es un conjunto de pares ordenados que relaciona
dos vértices.
   Vértice i ∈ V(G) ;  donde i es el vértice de inicio

   Vértice d ∈ V(G) ;  donde d es el vértice de destino objetivo

Salida:
   l(v) ∀v ∈ V − {i} ,  donde l(v) es el tiempo mínimo de llegar a v desde i.
   p(v) ∀v ∈ V − {i} ,  donde p(v) es el predecesor de v.

1 establecer
  l(i)=0
  M = Número muy grande
  R= ∅
  tripleta(h,v,g,k), tripleta de ventanas de tiempo k para el nodo v donde
el                  predecesor es h y el destino es g

  a(v,k)= límite inferior de la ventana de tiempo k para el nodo v
  b(v,k)= límite inferior de la ventana de tiempo k para el nodo v

2 Para todo v ∈ V-{i] hacer
  l(v)=M


3 "Encontrar un vértice v donde v ∈ V(G) y v∉ ℝ; tal que l(v)= min l(a)
donde   a ∈ V(G) y a∉R.

4 definir R=R U {V}

5 Para todo a donde ∈ V(G) Y a∉R; tal que (v,a)∈ E(G) hacer;
6   Si v≠i y v≠d entonces
7     Si a(v,k) ≤ l(v) ≤ b(v,k) entonces
8       Si l(a) > l(v) + w(v,a) entonces
          definir
          l(a)=l(v)+w(v,a)
          p(a)=v
7     Caso contrario;
        l(v)=a(v,k) donde k es la primera ventana de tiempo que se encuentra
la tripleta (h,v,a)
8       Si l(a)> l(v)+w(v,a) entonces
          definir
          l(a)=l(v) +w(v,a)
          p(a)=v


6   Caso contrario
7     si l(a)>l(v) * w(v,a) entonces
        definir
        l(a)= l(v) + w(v,a)
        p(a)=v
```

20 Figura 3.6. Pseudocódigo de Dijkstra modificado.

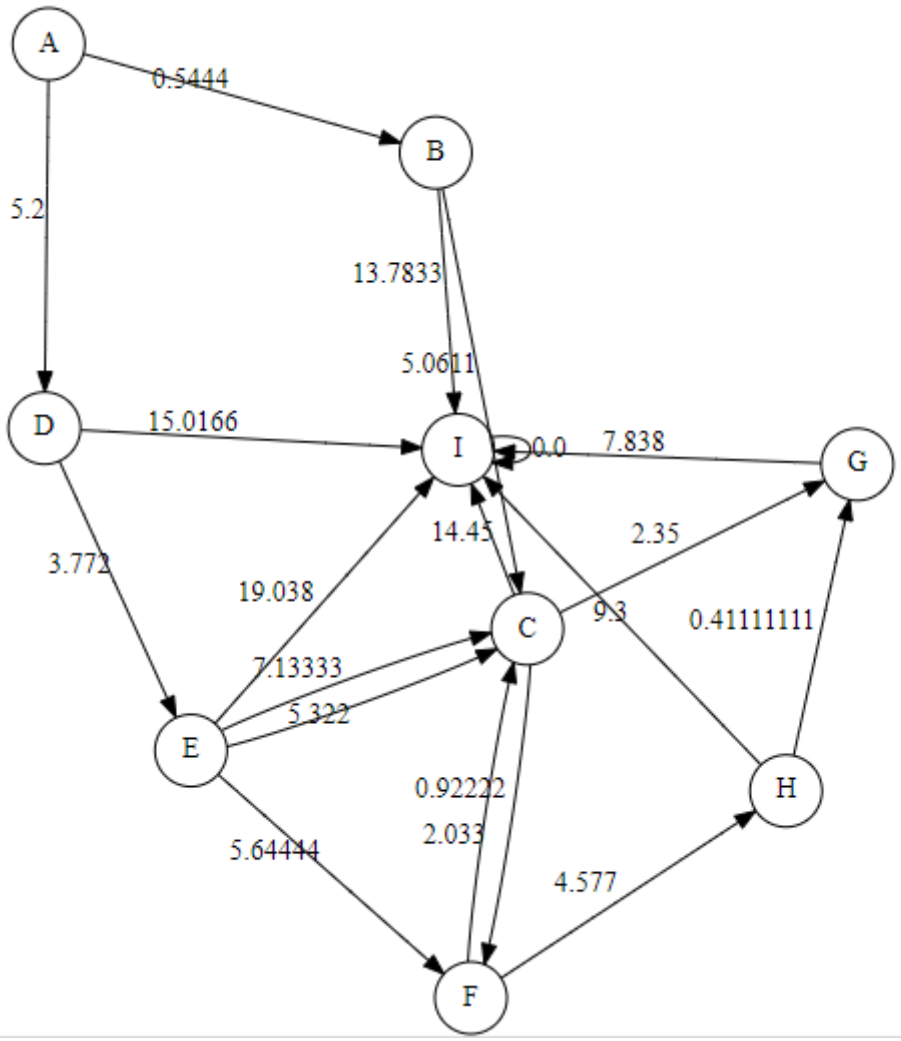
3.5 Pseudocódigo de Bellman Floyd modificado

```
Entrada:
   Dígrafo G,          Grafo a estudiar.
   Pesos w:E(G)→ℝ+;   E(G)  es un conjunto de pares ordenados que relaciona
dos vértices.
   Vértice i ∈V(G) ;  donde i es el vértice de inicio

   Vértice d ∈V(G) ;  donde d es el vértice de destino objetivo

Salida:
   l(v) ∀v∈V−{i} ,   donde l(v) es el tiempo mínimo de llegar a v desde i.
   p(v) ∀v∈V−{i} ,   donde p(v) es el predecesor de v.

1 establecer
  l(i)=0
  M = Número muy grande
  R= ∅
  tripleta(h,v,g,k), tripleta de ventanas de tiempo k para el nodo v donde
el                  predecesor es h y el destino es g

  a(v,k)= límite inferior de la ventana de tiempo k para el nodo v
  b(v,k)= límite inferior de la ventana de tiempo k para el nodo v

2 Para todo v ∈ V-{i] hacer
  l(v)=M


3 "Encontrar un vértice v donde v ∈ V(G) y v∉ ℝ; tal que l(v)= min l(a)
donde   a ∈ V(G) y a∉R.

4 definir R=R U {V}

5 Para todo i=1 hasta V-1
6   Para todo a donde ∈ V(G) Y a∉R; tal que (v,a)∈ E(G) hacer;
7     Si v≠i y v≠d entonces
8       Si a(v,k) ≤ l(v) ≤ b(v,k) entonces
9         Si l(a) >l(v)+w(v,a) entonces
            definir
            l(a)=l(v)+w(v,a)
            p(a)=v
8       Caso contrario;
          l(v)=a(v,k) donde k es la primera ventana de tiempo que se
encuentra la tripleta (h,v,a)
9         Si l(a)> l(v)+w(v,a) entonces
            definir
            l(a)=l(v) +w(v,a)
            p(a)=v


7     Caso contrario
8       si l(a)>l(v) * w(v,a) entonces
         definir
         l(a)= l(v) + w(v,a)
         p(a)=v

Si R∉ V(G) entonces volver al 3

Para todo a ∈V(G) y a ∉R; tal que (v,a) ∈E(G)
 Si l(a)>l(v)+w(v,a) entonces
  escribir tiene ciclo negativo
  parar
```

21Figura 3.7. Pseudocódigo de Bellman Floyd modificado.

Las diferencias que posee el algoritmo de Dijkstra y Bellman Ford se pueden observar en los pseudocódigos planteados. En el pseudocódigo de Dijkstra **Figura 3.6** en el **paso 5**, se tiene un bucle el cuál es el que realiza la evaluación y va obteniendo las distancias mínimas recorrida o distancias de salidas por cada uno de los nodos, mientras que para el algoritmo de Bellman Ford **figura 3.7** se tiene un bucle antes del bucle de Dijkstra **paso 5**, el cual se recorre $V-1$ veces lo que habíamos mencionado en el bucle de Dijkstra y es el que permite obtener un camino óptimo teniendo en cuenta la posibilidad de valores negativos en los pesos de las aristas. Además, en el algoritmo de Bellman después de culminar con el bucle del **paso 5,** se realiza la inspección de si el grafo posee ciclos negativos y que se puede observar en la parte inferior de la **figura 3.7**

3.6 Grafo.



22 Figura 3.8. Grafo obtenido por el código de programación Fuente: Varas & Alvarado

3.7 Modelo de programación lineal

A continuación, en la **figura 3.9** se puede observar el modelo de programación lineal utilizado para la implementación del diseño en GAMS (Cabezas, 2018). En este modelo se realizó una modificación en las restricciones y que se pueden observar en la **figura 3.9** sombreado de amarillo, estás fueron necesaria debido a que los nodos de origen y destino no poseen ventanas de tiempo.

Cabe destacar que se utilizará la misma información que se utilizó en los códigos de programación para la obtención de resultados y que fueron detallados en el capítulo 2.

Conjuntos:

V: *Conjuntos de nodos G ,* $\{s, v_1, v_2, \dots, d\}$ *, donde i es el nodo origen y d es el nodo destino*

A : *Conjuntos del arcos G,*

Θ_i: *Conjuntos del ventanas de tiempo de nodo i,* $\{wsi_i, w_{i1}, w_{i2}, \dots, w_{in}\}$ *, donde* wsi_i *es la primera ventana de tiem*

$PRED_i^k : \{h : (h, i, g) \in N_i^k\}$

$SUCC_i^k : \{g : (h, i, g) \in N_i^k\}$

Parametros:

a_i^k, b_i^k : *límite inferior y superior de la ventana de tiempo k del nodo i*

t_{ij} : *tiempo de ruta de ir del nodo i al j*

Variables:

$$x_{ij} = \begin{cases} 1, & \text{si el nodo } i \text{ es visitado por el nodo } j \\ 0, & \text{caso contrario} \end{cases}$$

$$y_i^k = \begin{cases} 1, & \text{si la ventana de tiempo } k \text{ es } \in \Theta_i \text{ es tomado en cuenta para el nodo } i \\ 0, & \text{caso contrario} \end{cases}$$

$T(i)$: *Tiempo de salida del nodo i*

Minimizar: $T(d)$

Restricciones:

$$\sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1 & i = i \\ 0 & i \in V \setminus \{s, d\} \\ -1 & i = t \end{cases} \qquad \forall i \in V$$

$$T_i + t_{ij} - T_j \leq M(1 - x_{ij}) \qquad \forall (i,j) \in A$$

$$a_i^k y_i^k \leq T_i \leq b_i^k + M(1 - y_i^k) \qquad \forall i \in V \setminus \{s, d\}, \forall k \in \Theta_i$$

$$y_i^k - \sum_{j \in PRED_i^k} x_{ji} \leq 0 \qquad \forall i \in V \setminus \{s\},\ \forall k \in \Theta_i$$

$$y_i^k - \sum_{j \in SUCC_i^k} x_{ij} \leq 0 \qquad \forall i \in V \setminus \{d\},\ \forall k \in \Theta_i$$

$$x_{ij} - \sum_{k \in \Theta_j} y_j^k \leq 0 \qquad \forall (i,j) \in A, j \neq \{d\}$$

$$x_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A$$

$$y_i^k \in \{0,1\}, \qquad \forall (i,j) \in A$$

$$T(i) \in \mathbb{R}^+ \qquad \forall i \in V$$

23 Figura 3.9. Modelo de programación lineal. Fuente: Xavier Cabezas

3.7.1 Resultados obtenidos del modelo de programación lineal.

```
---- EQU res9            .          .          .         1.000

                       LOWER      LEVEL      UPPER    MARGINAL

---- VAR S               -INF      14.328      +INF         .

---- VAR x

       LOWER      LEVEL      UPPER    MARGINAL

A.B        .      1.000      1.000 5.0000E+5
A.D        .        .        1.000       EPS
B.C        .        .        1.000       EPS
B.I        .      1.000      1.000 5.0000E+5
C.F        .        .        1.000       EPS
C.G        .        .        1.000       EPS
C.I        .        .        1.000       EPS
D.E        .        .        1.000       EPS
D.I        .        .        1.000       EPS
E.C        .        .        1.000       EPS
E.F        .        .        1.000       EPS
E.I        .        .        1.000       EPS
F.C        .        .        1.000       EPS
F.H        .        .        1.000       EPS
G.I        .        .        1.000       EPS
H.G        .        .        1.000       EPS
H.I        .        .        1.000       EPS
```

24 Figura 3.10. Resultado del diseño de programación lineal modelado en GAMS. Fuente: Varas & Alvarado

Variable $x(v_i, v_j)$ *si el nodo v_i es visitado por el nodo v_j*	
Variable	**Resultado**
$x(i, v_1), x(v_1, d)$	1
otros	0

Tabla 13 Resultado del diseño de programación lineal modelado en GAMS. Fuente: Varas & Alvarado

```
---- VAR ti

     LOWER      LEVEL      UPPER    MARGINAL

A     -INF        .         +INF        .
B     -INF      0.544       +INF        .
C     -INF        .         +INF        .
D     -INF        .         +INF        .
E     -INF        .         +INF        .
F     -INF        .         +INF        .
G     -INF        .         +INF        .
H     -INF        .         +INF        .
I     -INF     14.328       +INF        .

---- VAR y

          LOWER      LEVEL      UPPER    MARGINAL

B.v1        .         1.000      1.000       EPS
B.v2        .           .        1.000       EPS
B.v3        .           .        1.000       EPS
B.v4        .           .        1.000       EPS
B.v5        .           .        1.000       EPS
B.v6        .           .        1.000       EPS
B.v7        .           .        1.000       EPS
B.v8        .           .        1.000       EPS
B.v9        .           .        1.000       EPS
```

25 Figura 3.11. Resultado del diseño de programación lineal modelado en GAMS. Fuente: Varas & Alvarado

Variable $y(v_i, ws_{v_{i,k}})$ *si la ventana k es tomada en cuenta para el nodo* v_i	
Variable	**Resultado**
$y(v_1, ws_{v_{1,1}})$	1
otros	0

Tabla 14 Resultado del diseño de programación lineal modelado en GAMS. Fuente: Varas & Alvarado

Variable t(v_i) *tiempo óptimo de salida del nodo* v_i	
Variable	**Resultado (Minutos)**
$t(v_1)$	0,5444
$t(d)$	14,328
otros	0

Tabla 15 Resultado del diseño de programación lineal modelado en GAMS. Fuente: Varas & Alvarado

Variable t(v_i) *tiempo óptimo de salida del nodo* v_i	
Variable	**Resultado (Minutos)**
$t(v_1)$	0,5444
$t(d)$	14,328
otros	0

3.7.2 Comparación de resultado entre el Modelo de programación lineal y Heurísticas

En los resultados obtenidos por parte de la implementación del modelo de programación lineal en GAMS, se obtiene los resultados de las variables de decisión los cuales representan, el camino más corto entre ir del nodo origen i al nodo destino d, las ventanas de tiempos que se toman en cuenta en este camino y el tiempo óptimo que

tarda de ir del nodo i a d. Además, se puede determinar los predecesores del destino d al momento de obtener el camino más corto mencionado anteriormente. Esto se puede observar en la tabla de la **sección 3.7.1** al implementar el modelo de programación lineal se tiene solo el resultado del camino óptimo de los nodos que forman parte del camino más corto de ir de i a d, en comparación con el código de programación que nos muestra el camino óptimo de todos los nodos del grafo conjunto con sus predecesores y que se presenta como una mejor opción si se quiere tener una visión más allá de lo que se quiere.

CAPÍTULO 4

4. Conclusiones y Recomendaciones

4.1 Conclusiones

En el proyecto se pudo observar cómo se puede plantear correctamente estás restricciones de semaforización, pero cómo conclusión se debe destacar la importancia de encontrar un escenario en el cual se tome en cuenta todos los lineamientos para plantear correctamente las ventanas de tiempos, como por ejemplo, que en las fases de los semáforos de cada una de la intersecciones se tome en cuenta todos los nodos de un grafo, encontrar este escenario fue uno de los puntos más complejos al momento de encontrar un problema que podamos tomar como ejemplo para realizar un análisis, debido a esto se modificó el escenario actual real, ejemplo, una intersección que tenía 4 fases pasó a tener 2 fases.

Otro aspecto importante que se puede destacar a la hora de plantear esta problemática es la complejidad para obtener los datos, en específico el inicio de las ventanas de tiempo, en nuestro caso se lo realizó de manera aleatoria por motivos de que se necesita un equipo humano para hacer la toma de tiempo de este valor en todas las intersecciones en el mismo instante, es por esto por lo que este sería uno de los desafíos de quienes pueden hacer uso de nuestros algoritmos de solución propuestos. Asimismo, se necesitará de un gran proceso logístico y de campo en la obtención de la información de las aristas del grafo que en nuestro problema corresponden a los tiempos de circular de un nodo a otro nodo, donde hay que recalcar que esto se debe de realizar en una franja horaria y de días específicos, ya que se sabe que existe distintos tiempos en ir de un lugar a otro dependiendo de los horarios y días de circulación.

En cuanto a los resultados que se pueden obtener planteando las restricciones de semaforización, se puede intuir que este es un mejor planteamiento del problema si se quiere tomar intersecciones como nodos, debido a que en la situación actual no se toma en consideración estas variantes de semaforización y solo se toma en cuenta las distancias o el tiempo que existe entre circular de un nodo a otro. Esto puede alejar al investigador de obtener una solución óptima debido, a que se puede llegar a tener tiempos considerables que se omiten por no tomar en cuenta los controles de semaforización de las intersecciones.

En nuestro problema establecido en el cual se obtuvieron los resultados, se pudo identificar que existe una variación en el tiempo óptimo de la mitad de los nodos si se toma en cuenta la fase del semáforo. Podemos hablar de que existe una variación en los nodos que es muy pequeña, pero esto se debió a varias situaciones del problema planteado, como lo es, que el problema se considera pequeño debido a las pocos nodos o intersecciones que tiene el grafo y los grados que tienen cada uno de ellos que no supera la cantidad de 4 y que en sí representa la adyacencia que existe con otros nodos. Además, de algo importante como lo es las fases de las intersecciones, ya que se sabe que entre más fases tenga un semáforo mayor puede llegar a ser el valor del tiempo de salida de un nodo a otro, lo que incrementaría el valor de tiempo óptimo de llegar del punto de origen a los distintos nodos.

4.2 Recomendaciones

Se recomienda tener una visión clara de la problemática antes de tomar en cuenta los algoritmos de este proyecto, esto con el objetivo de evitar tratar plantear un problema el cuál se resuelva por otro tipo de algoritmo. De igual manera, se requiere tener esta visión clara para poder plantear un correcto grafo, comenzando por identificar cuáles son los nodos ó intersecciones que se involucrarán cómo parte del problema, se sabe que existe intersecciones que no poseen semáforos por lo que no podrían formar parte del grafo que se requiere plantear. Asimismo, tener claro de las vías, avenidas o calles que se involucrarán al momento de querer definir las aristas del grafo.

En cuanto, a la información levantada se necesita tener un gran equipo que permita obtenerla, cómo se habló en gran parte del proyecto, existe datos que la complejidad de poseer la información precisa radica en la cantidad de personas necesarias para realizar la captación de esta, y esta complejidad aumentará dependiendo del problema que se quiere resolver, es decir si hablamos de un grafo que va obtener una gran cantidad de

nodos o intersecciones y aristas, se habla de un proceso logístico arduo que llevará un considerable tiempo y esfuerzo humano.

Por otra parte, este proyecto fomenta la investigación a nuevas consideraciones del problema, por ejemplo, una de las modificaciones sería la necesidad que ciertos nodos del grafo formen parte del camino más corto, es decir, que obligatoriamente sean visitados ó también que el problema considere que algunos nodos no contengan restricciones de semaforización para así tomar en cuenta más todas las intersecciones de una red. Así mismo, en el modelo de programación lineal se puede modificar la función objetivo, en donde nuestra función objetivo sea obtener la ruta con menos tiempos de espera en las intersecciones.

Bibliografías

Aguilar, J. (26 de 02 de 2021). *Analisis de riego.* Obtenido de https://www.coursehero.com/file/93574890/analisis-de-riegodocx/

Arruabarrena, R. (16 de 03 de 2000). *66 problemas resueltos de.* Obtenido de http://creativecommons.org/licenses/by-nc-sa/3.0/es/

Bellman, R. (1958). *Scirp.* Obtenido de http://www.scirp.org/(S(i43dyn45teexjx455qlt3d2q))/reference/Reference

Blanco, M. (2019). *Guía de introducción.* Obtenido de file:///C:/Users/joddel.varas/Downloads/Gams_tutorial_2019.pdf

Cabezas, X. (2018). *Methods for Solving Two Discrete Optimization Problems.* The University of Edinburgh, Edinburgh, UK.

Chávez, M. A. (2014). *Inventio, la génesis de la cultura universitaria en Morelos.* México.

Jiang Jin & Chen. (2013). Finding the K shortest paths in a time-schedule network. En S. C. Wen Jin, *Computers & Operations Research* (págs. 2975-2982). Beijing.

Lozano, S. &. (2001). *Algoritmo de Dijkstra.* Universidad Politécnica de Madrid, Facultad de Informática, Madrid. Obtenido de http://bioinfo.uib.es/~joemiro/aenui/procJenui/ProcWeb/actas2001/saalg223.pdf

Lucca, A. M. (2015). *Teoría de Grafos.* Universidad Nacional de la Patagonia San Juan Bosco, Departamento de Matemática., Comodoro Rivadavia. Obtenido de https://www.studocu.com/es-mx/document/universidad-abierta-y-a-distancia-de-mexico/contaduria-y-finanzas-pblicas/10926-texto-del-articulo-28704-1-10-2015-0416/16537772

Safe, M. D. (2018). *Árboles y distancia.* Técnicos, Instituto de Cálculo de Universidad de Buenos Aires, Buenos Aires. Obtenido de https://www.ic.fcen.uba.ar/materias/grafosSafe/Arboles.pdf

Santibañez, M. d. (2012). *Características básicas de los problemas de Programación lineal.* Tecnico, Madrid. Obtenido de https://xdoc.mx/documents/caracteristicas-basicas-de-los-problemas-de-programacion-lineal-5e518def92eb3

Yen-Liang & Yang. (2000). Shortest paths in traffic-light networks. En Y.-L. Chen, *Transportation Research Part B: Methodological* (Vol. vol. 34(4), págs. 241-253). Taiwan.

Anexos

Código de programación de los 4 algoritmos (Dijkstra, Bellman, Dijkstra modificado, Bellman modificado)

Instalar las librerias

```
!pip install graphviz
```

Importar las librerias necesarias

```
[1] from graphviz import Digraph
    import pandas as pd
    import os
    os.environ["PATH"] += os.pathsep + 'D:/Program Files (x86)/Graphviz2.38/bin/'
```

Crear la funcion para mosrar el grafo

```
def grafo(datos):
    #Esta clase crea la visualizacion del grafo

    G = Digraph(edge_attr={'len':'2.5'})
    G.attr(rankdir='LR', size='8')
    G.attr('node', shape='circle')
    G.attr(layout="neato")
    nodelist = []

    for idx, row in df.iterrows():

        node1, node2, weight = [str(i) for i in row]
        if node1 not in nodelist:
            G.node(node1)
            nodelist.append(node2)
        if node2 not in nodelist:
            G.node(node2)
            nodelist.append(node2)

        G.edge(node1,node2, label = weight)

    return G
```

Importar los datos del csv

```
[3] df = pd.read_csv('Libro1.csv', header=None)
```

Crear el grafo

```
[4] grafo1=grafo(df)
    grafo1
```

Creacion de funcion para pasar de un dataframe a un diccionario

```
[5] def df_dic(df):
        #Esta funcion transforma el dataframe en un diccionario con la estructura adecuada para ser utilizada despues
        dicc={}
        nodos=[]
        for i in df.iterrows():
            if(i[1][0] not in nodos):
                nodos.append(i[1][0])
                dicc[i[1][0]]={}

            dicc[i[1][0]][i[1][1]]=i[1][2]

        return dicc,nodos
```

```
[6] dicc,nodos=df_dic(df)
    dicc
```

```
{'A': {'B': 0.5444, 'D': 5.2},
 'B': {'C': 5.0611, 'I': 13.7833},
 'C': {'G': 2.35, 'I': 14.45, 'F': 0.92222},
 'D': {'I': 15.0166, 'E': 3.772},
 'E': {'C': 5.322, 'I': 19.038, 'F': 5.64444},
 'F': {'C': 2.033, 'H': 4.577},
 'G': {'I': 7.838},
 'H': {'I': 9.3, 'G': 0.41111111},
 'I': {'I': 0.0}}
```

```
        conteo=0
        # Relajación
        for _ in range(self.V - 1):
           conteo1=conteo1+1

           for s, d, w in self.graph:
                conteo=conteo+1
                #print(conteo)
                #print(s,d,w)
                if dist[s] != float("Inf") and dist[s] + w < dist[d]:
                    dist[d] = dist[s] + w
                    predesesores[d]= s

        # Detectar ciclo negativo
        # si el valor cambia entonces tenemos un ciclo negativo en el gráfico
        # y no podemos encontrar las distancias más cortas
        for s, d, w in self.graph:
            if dist[s] != float("Inf") and dist[s] + w < dist[d]:
                print("El gráfico contiene un ciclo de peso negativo")
                return
        print(predesesores)
        print("número de iteraciones Dijkstra",conteo1)
        print("numero iteraciones Bellman",conteo -(conteo1))
        #¡No se encontró ningún ciclo de peso negativo!
        # Imprime la matriz de distancia y predecesora
        self.print_solution(dist)
```


```
        conteo=0
        # Relajación
        for _ in range(self.V - 1):
           conteo1=conteo1+1

           for s, d, w in self.graph:
                conteo=conteo+1
                #print(conteo)
                #print(s,d,w)
                if dist[s] != float("Inf") and dist[s] + w < dist[d]:
                    dist[d] = dist[s] + w
                    predesesores[d]= s

        # Detectar ciclo negativo
        # si el valor cambia entonces tenemos un ciclo negativo en el gráfico
        # y no podemos encontrar las distancias más cortas
        for s, d, w in self.graph:
            if dist[s] != float("Inf") and dist[s] + w < dist[d]:
                print("El gráfico contiene un ciclo de peso negativo")
                return
        print(predesesores)
        print("número de iteraciones Dijkstra",conteo1)
        print("numero iteraciones Bellman",conteo -(conteo1))
        #¡No se encontró ningún ciclo de peso negativo!
        # Imprime la matriz de distancia y predecesora
        self.print_solution(dist)
```


```
[8]  def constructBellmanCollection(data, source):
         graph = Graph(len(data))
         listado=range(0,len(data))
         listadoReal=[]
         for i in data:
             listadoReal.append(i)
         for i in data:
             for e in data[i]:
                 iIndex=listadoReal.index(i)
                 eIndex = listadoReal.index(e)
                 graph.add_edge(listado[iIndex], listado[eIndex], data[i][e])

         dist = graph.bellman_ford(listado[listadoReal.index("A")])

         print("Vertex Distance from Source")
         for i in range(graph.V):
             iIndex = listado.index(i)
             print(listadoReal[iIndex], "es nodo", i)


     constructBellmanCollection(dicc, "A")
```

```
[inf, 0, 1, 0, 3, 2, 2, 5, 1]
número de iteraciones Dijkstra 8
numero iteraciones Bellman 136
Distancia del vértice desde la fuente
0               0
1               0.5444
2               5.605499999999999
3               5.2
4               8.972
5               6.5277199999999995
6               7.955499999999999
7               11.10472
8               14.3277
Vertex Distance from Source
A es nodo 0
B es nodo 1
C es nodo 2
D es nodo 3
E es nodo 4
F es nodo 5
G es nodo 6
H es nodo 7
I es nodo 8
```

▾ Algoritmo de Dijkstra

```
[10]
    class Graph:

        def __init__(self, vertices):
            self.V = vertices   # Número total de vértices en el gráfico
            self.graph = []     # Array de aristas

        # Add aristas
        def add_edge(self, s, d, w):
            self.graph.append([s, d, w])

        # Imprimir solucion
        def print_solution(self, dist):
            print("Distancia del vértice desde la fuente")
            for i in range(self.V):
                print("{0}\t\t{1}".format(i, dist[i]))

        def Dijkstra(self, src):

            #Completar la matriz de distancia y la matriz predecesora
            dist = [float("Inf")] * self.V

            predesesores=[float("Inf")] * self.V

            dist[src] = 0
            conteo1=0
            conteo=0
            for s, d, w in self.graph:
```

```
            for s, d, w in self.graph:
                    conteo=conteo+1
                    #print(conteo)
                    #print(s,d,w)
                    if dist[s] != float("Inf") and dist[s] + w < dist[d]:
                        dist[d] = dist[s] + w
                        predesesores[d]= s


            self.print_solution(dist)
```

```
            self.print_solution(dist)
```

```
def constructBellmanCollection(data, source):
    graph = Graph(len(data))
    listado=range(0,len(data))
    listadoReal=[]
    for i in data:
        listadoReal.append(i)
    for i in data:
        for e in data[i]:
            iIndex=listadoReal.index(i)
            eIndex = listadoReal.index(e)
            graph.add_edge(listado[iIndex], listado[eIndex], data[i][e])

    dist = graph.Dijkstra(listado[listadoReal.index("A")])

    print("Distancia mas")
    for i in range(graph.V):
        iIndex = listado.index(i)
        print(listadoReal[iIndex], "es nodo", i)


constructBellmanCollection(dicc, "A")
```

```
        iIndex = listado.index(i)
        print(listadoReal[iIndex], "es nodo", i)


constructBellmanCollection(dicc, "A")
```

```
Distancia del vértice desde la fuente
0               0
1               0.5444
2               5.605499999999999
3               5.2
4               8.972
5               6.527719999999995
6               7.955499999999999
7               11.10472
8               14.3277
Distancia mas
A es nodo 0
B es nodo 1
C es nodo 2
D es nodo 3
E es nodo 4
F es nodo 5
G es nodo 6
H es nodo 7
I es nodo 8
```

```
[14] dicc_nodos={0:'A', 1:'B', 2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',6:'I'}
```

```
ventanas_dicc={'B': {'A':{'C': 1,'I':1, 'D': 2 }},
'C': {'B':{'I': 2 ,'G': 1},'E':{'G':1, 'F': 1}},
'D': {'A':{'E': 1,'C': 1,'I': 2}},
'E': {'D':{'A': 1,'B': 1,'F': 2}},
'F': {'D':{'C': 1},'C':{'H': 2}},
'G': {'C':{'I': 1},'H':{'I': 2}},
'H': {'F':{'I': 1},'F':{'G': 2}}}

print(ventanas_dicc)
```

```
{'B': {'A': {'C': 1, 'I': 1, 'D': 2}}, 'C': {'B': {'I': 2, 'G': 1}, 'E': {'G': 1, 'F': 1}}, 'D': {'A': {'E': 1, 'C': 1, 'I': 2}}, 'E': {'D': {'A': 1, 'B': 1, 'F': 2}}, 'F': {'D': {'
```

Creación de las ventas de tiempo

```
lista_distancia=[[0.12,0.57,0.75],[0.23,0.72,0.67],[0.21,0.63,0.67],[0.07,0.75,0.58],[0.32,0.92,0.75],[0.18,0.72,0.67],[0.43,0.58,0.58]]
lista_nodos= nodos
print(lista_nodos)
ventanas=[]
for k in range(0,len(lista_nodos)):
  ventanas.append([])

for k in range(1,len(lista_nodos)-1):
   for j in range (0,10):
      if j==0:
       superior=lista_distancia[k-1][0]
       ventanas[k-1].append(round(superior,5))

      else:
       for t in range(0,len(lista_distancia[k-1])-1):

         superior= superior + lista_distancia[k-1][t+1]

         ventanas[k-1].append(round(superior,5))

ventanas_hechas=[]
for k in range(0,len(lista_nodos)-2):
  ventanas_hechas.append([])
for k in range(0,len(lista_nodos)-2):
  for p in range(0,len(ventanas[k])-1):
    ventanas_hechas[k].append([ventanas[k][p],ventanas[k][p+1]])
```

```
print(ventanas_hechas)

dic_ventan_nodos={}
for k in range(1,len(lista_nodos)-1):
  dic_ventan_nodos[lista_nodos[k]]=ventanas_hechas[k-1]
print(dic_ventan_nodos)
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
[[[0.12, 0.69], [0.69, 1.44], [1.44, 2.01], [2.01, 2.76], [2.76, 3.33], [3.33, 4.08], [4.08, 4.65], [4.65, 5.4], [5.4, 5.97], [5.97, 6.72], [6.72, 7.29], [7.29, 8.04], [8.04, 8.61],
{'B': [[0.12, 0.69], [0.69, 1.44], [1.44, 2.01], [2.01, 2.76], [2.76, 3.33], [3.33, 4.08], [4.08, 4.65], [4.65, 5.4], [5.4, 5.97], [5.97, 6.72], [6.72, 7.29], [7.29, 8.04], [8.04, 8.
```

ASIGNAR EL NÚMERO DE VENTANAS DE TIEMPOS

```
[ ] numeros_venta_tiempo=[2,2,2,2,2,2,2]
    lista_numeros_ventanas=[]

    for k in range(0,len(numeros_venta_tiempo)):
      lista_numeros_ventanas.append([])
      for i in range(0,len(dic_ventan_nodos[list(dic_ventan_nodos)[k]])):
        for t in range(1,numeros_venta_tiempo[k]+1):
          lista_numeros_ventanas[k].append(t)
    print(lista_numeros_ventanas)
```

```
[[1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2], [1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2
```

Código de Bellman modificado

```
[24] class Graph:
        def __init__(self, vertices):
            self.V = vertices
            self.graph = []


        def add_edge(self, s, d, w):
            self.graph.append([s, d, w])

        def print_solution(self, dist):
            print("Distancia del vértice desde la fuente")
            for i in range(self.V):
                print("{0}\t\t{1}".format(i, dist[i]))

        def proyecto_codigo(self, src,dic_ventan_nodos,ventanas_dicc,dicc_nodos,lista_numeros_ventanas):


            dista = [float("Inf")] * self.V

            dista[src] = 0
            predesesores=[float("Inf")] * self.V

            for _ in range(self.V - 1):
              for s, d, w in self.graph:
                 if (dista[s] != float("Inf")):
                    salida = dista[s]
```

```
[24]
            dista[src] = 0
            predesesores=[float("Inf")] * self.V

            for _ in range(self.V - 1):
              for s, d, w in self.graph:
                 if (dista[s] != float("Inf")):
                    salida = dista[s]
                    posicion=0
                    if s>0 and s<5:
                      for i in range(0,len(dic_ventan_nodos[list(dic_ventan_nodos)[s-1]])): #len(dic_ventan_nodos[list(dic_ventan_nodos)[s-1]])-- cantidad de pares de ventanas de tiempo
                        if i>0 and i<len(dic_ventan_nodos[list(dic_ventan_nodos)[s-1]]):
                          if salida>=dic_ventan_nodos[list(dic_ventan_nodos)[s-1]][i-1][0] and salida<=dic_ventan_nodos[list(dic_ventan_nodos)[s-1]][i-1][1]: #condición de intervalo
                           posicion = i

                          if posicion!=lista_numeros_ventanas[s][posicion+1]:
                             salida=dic_ventan_nodos[dicc_nodos[s]][i-1][1]

                             break

                    if salida != float("Inf") and salida + w < dista[d]:
                      #print(salida,"+", w)
                      dista[d] = salida + w

                      predesesores[d]= s
            print(predesesores)
            self.print_solution(dista)
```

```
[25] def prod_proyecto_codigo(data, source,dic_ventan_nodos,ventanas_hechas,dicc_nodos):
        graph = Graph(len(data))
        listado=range(0,len(data))
        listadoReal=[]
        for i in data:
            listadoReal.append(i)
        for i in data:
            for e in data[i]:
                iIndex=listadoReal.index(i)
                eIndex = listadoReal.index(e)
                graph.add_edge(listado[iIndex], listado[eIndex], data[i][e])

        dist = graph.proyecto_codigo(listado[listadoReal.index("A")],dic_ventan_nodos,ventanas_dicc,dicc_nodos,lista_numeros_ventanas)

        print("Distancia del vértice desde la fuente")
        for i in range(graph.V):
            iIndex = listado.index(i)
            print(listadoReal[iIndex], "es nodo", i)


    prod_proyecto_codigo(dicc, "s",dic_ventan_nodos,ventanas_hechas,dicc_nodos)
```

Código de Dijkstra modificado

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []


    def add_edge(self, s, d, w):
        self.graph.append([s, d, w])

    def print_solution(self, dist):
        print("Distancia del vértice desde la fuente")
        for i in range(self.V):
            print("{0}\t\t{1}".format(i, dist[i]))

    def proyecto_codigo_Dijkstra(self, src,dic_ventan_nodos,ventanas_dicc,dicc_nodos,lista_numeros_ventanas):


        dista = [float("Inf")] * self.V

        dista[src] = 0
        predesesores=[float("Inf")] * self.V

        for s, d, w in self.graph:
             if (dista[s] != float("Inf")):
                salida = dista[s]
                posicion=0
                if s>0 and s<5:
                  for i in range(0,len(dic_ventan_nodos[list(dic_ventan_nodos)[s-1]])): #len(dic_ventan_nodos[list(dic_ventan_nodos)[s-1]])-- cantidad de pares de ventanas de tiempo
                    if i>0 and i<len(dic_ventan_nodos[list(dic_ventan_nodos)[s-1]]):
```

```
            for s, d, w in self.graph:
                 if (dista[s] != float("Inf")):
                    salida = dista[s]
                    posicion=0
                    if s>0 and s<5:
                      for i in range(0,len(dic_ventan_nodos[list(dic_ventan_nodos)[s-1]])): #len(dic_ventan_nodos[list(dic_ventan_nodos)[s-1]])-- cantidad de pares de ventanas de tiempo
                        if i>0 and i<len(dic_ventan_nodos[list(dic_ventan_nodos)[s-1]]):
                          if salida>=dic_ventan_nodos[list(dic_ventan_nodos)[s-1]][i-1][0] and salida<=dic_ventan_nodos[list(dic_ventan_nodos)[s-1]][i-1][1]: #condición de intervalo
                           posicion = i

                          if posicion!=lista_numeros_ventanas[s][posicion+1]:
                             salida=dic_ventan_nodos[dicc_nodos[s]][i-1][1]

                             break

                    if salida != float("Inf") and salida + w < dista[d]:
                      #print(salida,"+", w)
                      dista[d] = salida + w

                      predesesores[d]= s
            print(predesesores)
            self.print_solution(dista)
```

```python
def prod_proyecto_codigo(data, source,dic_ventan_nodos,ventanas_hechas,dicc_nodos):
    graph = Graph(len(data))
    listado=range(0,len(data))
    listadoReal=[]
    for i in data:
        listadoReal.append(i)
    for i in data:
        for e in data[i]:
            iIndex=listadoReal.index(i)
            eIndex = listadoReal.index(e)
            graph.add_edge(listado[iIndex], listado[eIndex], data[i][e])

    dist = graph.proyecto_codigo_Dijkstra(listado[listadoReal.index("A")],dic_ventan_nodos,ventanas_dicc,dicc_nodos,lista_numeros_ventanas)

    print("Distancia del vértice desde la fuente")
    for i in range(graph.V):
        iIndex = listado.index(i)
        print(listadoReal[iIndex], "es nodo", i)


prod_proyecto_codigo(dicc, "s",dic_ventan_nodos,ventanas_hechas,dicc_nodos)
```

```python
prod_proyecto_codigo(dicc, "s",dic_ventan_nodos,ventanas_hechas,dicc_nodos)
```

```
[inf, 0, 1, 0, 3, 2, 2, 5, 1]
Distancia del vértice desde la fuente
0               0
1               0.5444
2               5.605499999999999
3               5.2
4               9.182
5               6.71222
6               8.14
7               11.28922
8               14.3277
Distancia del vértice desde la fuente
A es nodo 0
B es nodo 1
C es nodo 2
D es nodo 3
E es nodo 4
F es nodo 5
G es nodo 6
H es nodo 7
I es nodo 8
```

Implementación del modelo de programación lineal en GAMS.

```
SET i indicador del origen del arco (nodo) /A,B,C,D,E,F,G,H,I/
SET k ventana de tiempo para el nodo /v1*v22/
conex(i,i) arco
/A.B,B.C,C.G,D.I,A.D,E.C,F.C,D.E,E.I,E.F,G.I,C.I,F.H,H.I,B.I,C.F,H.G/
vent(i,k) ventanas
/B.(v1*v22), C.(v1*v22), D.(v1*v22), E.(v1*v22), F.(v1*v22), G.(v1*v22), H.(v1*v22)/
tripletasuc (i,i,k) tripletas sucesores
/B.C.v1,B.I.v1,B.C.v2,B.I.v2,B.C.v3,B.I.v3,B.C.v4,B.I.v4,B.C.v5,B.I.v5,B.C.v6,B.I.v6,B.C.v7,B.I.v7,B.C.v8,B.I.v8,B.C.v9,B.I.v9,B.C.v10,B.I.v10,B.C.v11,B.I.v11,
C.G.v1,C.F.v1,C.I.v2,C.G.v3,C.F.v3,C.I.v4,C.G.v5,C.F.v5,C.I.v6,C.G.v7,C.F.v7,C.I.v8,C.G.v9,C.F.v9,C.I.v10,C.G.v11,C.F.v11,C.I.v12,C.G.v13,C.F.v13,C.I.v14,C.G.v
D.E.v1,D.I.v2,D.E.v3,D.I.v4,D.E.v5,D.I.v6,D.E.v7,D.I.v8,D.E.v9,D.I.v10,D.E.v11,D.I.v12,D.E.v13,D.I.v14,D.E.v15,D.I.v16,D.E.v17,D.I.v18,D.E.v19,D.I.v20,D.E.v21,
E.I.v1,E.F.v2,E.I.v3,E.F.v4,E.I.v5,E.F.v6,E.I.v7,E.F.v8,E.I.v9,E.F.v10,E.I.v11,E.F.v12,E.I.v13,E.F.v14,E.I.v15,E.F.v16,E.I.v17,E.F.v18,E.I.v19,E.F.v20,E.I.v21,
F.C.v1,F.H.v2,F.C.v3,F.H.v4,F.C.v5,F.H.v6,F.C.v7,F.H.v8,F.C.v9,F.H.v10,F.C.v11,F.H.v12,F.C.v13,F.H.v14,F.C.v15,F.H.v16,F.C.v17,F.H.v18,F.C.v19,F.H.v20,F.C.v21,
G.I.v1,G.I.v2,G.I.v3,G.I.v4,G.I.v5,G.I.v6,G.I.v7,G.I.v8,G.I.v9,G.I.v10,G.I.v11,G.I.v12,G.I.v13,G.I.v14,G.I.v15,G.I.v16,G.I.v17,G.I.v18,G.I.v19,G.I.v20,G.I.v21,
H.I.v1,H.G.v2,H.I.v3,H.G.v4,H.I.v5,H.G.v6,H.I.v7,H.G.v8,H.I.v9,H.G.v10,H.I.v11,H.G.v12,H.I.v13,H.G.v14,H.I.v15,H.G.v16,H.I.v17,H.G.v18,H.I.v19,H.G.v20,H.I.v21,
/

tripletapred (i,i,k)  tripletas predecesores
/A.B.v1,A.B.v2,A.B.v3,A.B.v4,A.B.v5,A.B.v6,A.B.v7,A.B.v8,A.B.v9,A.B.v10,A.B.v11,A.B.v12,A.B.v13,A.B.v14,A.B.v15,A.B.v16,A.B.v17,A.B.v18,A.B.v19,A.B.v20,A.B.v21
B.C.v1,E.C.v1,B.C.v2,B.C.v3,E.C.v3,B.C.v4,B.C.v5,E.C.v5,B.C.v6,B.C.v7,E.C.v7,B.C.v8,B.C.v9,E.C.v9,B.C.v10,B.C.v11,E.C.v11,B.C.v12,B.C.v13,E.C.v13,B.C.v14,B.C.v
A.D.v1,A.D.v2,A.D.v3,A.D.v4,A.D.v5,A.D.v6,A.D.v7,A.D.v8,A.D.v9,A.D.v10,A.D.v11,A.D.v12,A.D.v13,A.D.v14,A.D.v15,A.D.v16,A.D.v17,A.D.v18,A.D.v19,A.D.v20,A.D.v21,
D.E.v1,D.E.v2,D.E.v3,D.E.v4,D.E.v5,D.E.v6,D.E.v7,D.E.v8,D.E.v9,D.E.v10,D.E.v11,D.E.v12,D.E.v13,D.E.v14,D.E.v15,D.E.v16,D.E.v17,D.E.v18,D.E.v19,D.E.v20,D.E.v21,
E.F.v1,C.F.v2,E.F.v3,C.F.v4,E.F.v5,C.F.v6,E.F.v7,C.F.v8,E.F.v9,C.F.v10,E.F.v11,C.F.v12,E.F.v13,C.F.v14,E.F.v15,C.F.v16,E.F.v17,C.F.v18,E.F.v19,C.F.v20,E.F.v21,
C.G.v1,H.G.v2,C.G.v3,H.G.v4,C.G.v5,H.G.v6,C.G.v7,H.G.v8,C.G.v9,H.G.v10,C.G.v11,H.G.v12,C.G.v13,H.G.v14,C.G.v15,H.G.v16,C.G.v17,H.G.v18,C.G.v19,H.G.v20,C.G.v21,
F.H.v1,F.H.v2,F.H.v3,F.H.v4,F.H.v5,F.H.v6,F.H.v7,F.H.v8,F.H.v9,F.H.v10,F.H.v11,F.H.v12,F.H.v13,F.H.v14,F.H.v15,F.H.v16,F.H.v17,F.H.v18,F.H.v19,F.H.v20,F.H.v21,
```

```
alias ( i, j);

parameter
t(i,j)    tiempo asociado con el arco i j
/
A.B 0.5444
B.C 5.0611
C.G 2.35
D.I 15.0166
A.D 5.2
F.C 2.033
D.E 3.772
E.C 5.322
E.I 19.038
E.F 5.64444
G.I 7.838
C.I 14.45
F.H 4.577
H.I 9.3
B.I 13.7833
C.F 0.92222
H.G 0.41111111
/
table
a(i, k) limite inferior de la ventan de tiempo k en el nodo i
```

```
/
table
a(i, k) limite inferior de la ventan de tiempo k en el nodo i

    v1    v2    v3    v4    v5    v6    v7    v8    v9    v10   v11   v12   v13   v14   v15   v16   v17   v18   v19   v20   v21   v22
B  0.12  0.69  1.44  2.01  2.76  3.33  4.08  4.65  5.4   5.97  6.72  7.29  8.04  8.61  9.36  9.93  10.68 11.25 12.0  12.57 13.32 13.89
C  0.23  0.95  1.62  2.34  3.01  3.73  4.4   5.12  5.79  6.51  7.18  7.9   8.57  9.29  9.96  10.68 11.35 12.07 12.74 13.46 14.13 14.85
D  0.21  0.84  1.51  2.14  2.81  3.44  4.11  4.74  5.41  6.04  6.71  7.34  8.01  8.64  9.31  9.94  10.61 11.24 11.91 12.54 13.21 13.84
E  0.07  0.82  1.4   2.15  2.73  3.48  4.06  4.81  5.39  6.14  6.72  7.47  8.05  8.8   9.38  10.13 10.71 11.46 12.04 12.79 13.37 14.12
F  0.32  1.24  1.99  2.91  3.66  4.58  5.33  6.25  7     7.92  8.67  9.59  10.34 11.26 12.01 12.93 13.68 14.6  15.35 16.27 17.02 17.94
G  0.18  0.9   1.57  2.29  2.96  3.68  4.35  5.07  5.74  6.46  7.13  7.85  8.52  9.24  9.91  10.63 11.3  12.02 12.69 13.41 14.08 14.8
H  0.43  1.01  1.59  2.17  2.75  3.33  3.91  4.49  5.07  5.65  6.23  6.81  7.39  7.97  8.55  9.13  9.71  10.29 10.87 11.45 12.03 12.61

table
b(i, k) limite inferior de la ventan de tiempo k en el nodo i

    v1    v2    v3    v4    v5    v6    v7    v8    v9    v10   v11   v12   v13   v14   v15   v16   v17   v18   v19   v20   v21   v22
B  0.69  1.44  2.01  2.76  3.33  4.08  4.65  5.4   5.97  6.72  7.29  8.04  8.61  9.36  9.93  10.68 11.25 12.0  12.57 13.32 13.89 14.64
C  0.95  1.62  2.34  3.01  3.73  4.4   5.12  5.79  6.51  7.18  7.9   8.57  9.29  9.96  10.68 11.35 12.07 12.74 13.46 14.13 14.85 15.52
D  0.84  1.51  2.14  2.81  3.44  4.11  4.74  5.41  6.04  6.71  7.34  8.01  8.64  9.31  9.94  10.61 11.24 11.91 12.54 13.21 13.84 14.51
E  0.82  1.4   2.15  2.73  3.48  4.06  4.81  5.39  6.14  6.72  7.47  8.05  8.8   9.38  10.13 10.71 11.46 12.04 12.79 13.37 14.12 14.7
F  1.24  1.99  2.91  3.66  4.58  5.33  6.25  7     7.92  8.67  9.59  10.34 11.26 12.01 12.93 13.68 14.6  15.35 16.27 17.02 17.94 18.69
G  0.9   1.57  2.29  2.96  3.68  4.35  5.07  5.74  6.46  7.13  7.85  8.52  9.24  9.91  10.63 11.3  12.02 12.69 13.41 14.08 14.8  15.47
H  1.01  1.59  2.17  2.75  3.33  3.91  4.49  5.07  5.65  6.23  6.81  7.39  7.97  8.55  9.13  9.71  10.29 10.87 11.45 12.03 12.61 13.19
```

```
parameter
pp(i)
/
A      1
B      0
C      0
D      0
E      0
F      0
G      0
H      0
I     -1 /;
variables S, x(i,j), ti(i), y(i,k);
binary variables x(i,j),y(i,k);

equation
funobjetivo
res1(i)
res2(i,j)
res3(i,j)
res4(i,k)
res5(i,k)
res6(i,k)
res7(i,k)
res8(i,j)
res9
*res10
;
funobjetivo.. S =e=  ti('I');
res1(i).. pp(i) =e= + (sum(conex(i,j), x(i,j)))- sum(conex(j,i), x(j,i));
```

```
res4(i,k)
res5(i,k)
res6(i,k)
res7(i,k)
res8(i,j)
res9
*res10
;
funobjetivo.. S =e=  ti('I');
res1(i).. pp(i) =e= + (sum(conex(i,j), x(i,j)))- sum(conex(j,i), x(j,i));
res2(conex(i,j))$(ord(i)<2).. ti(i)+ t(i,j)- ti(j)=l= 500000 - 500000*x(i,j);
res3(conex(i,j))$(ord(i)>1).. ti(i)+ t(i,j)- ti(j)=l= +500000 - 500000*x(i,j) ;
res4(vent(i,k)) .. a(i,k)*y(i,k)=l= ti(i) ;
res5(vent(i,k)).. ti(i)=l= b(i,k)+(1000000*(1-y(i,k)));
res6(vent(i,k)).. y(i,k)- sum(tripletapred(j,i,k), x(j,i))=l= 0;
res7(vent(i,k)).. y(i,k)- sum(tripletasuc(i,j,k), x(i,j))=l= 0;
res8(conex(i,j))$(ord(i)<9 and ord(j)<9).. x(i,j)- sum(vent(j,k), y(j,k))=l= 0;
res9.. ti('A')=e=0;
*res10.. x('B','I')=e=1;


MODEL pruebacamino /all/;
SOLVE pruebacamino using MIP minimizing S
;
*(ord(i)>1) and(ord(i)<6)
```