ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

DESARROLLO DE UN ALGORITMO HÍBRIDO PARA RECOMENDACIONES DE ITINERARIOS TURÍSTICOS DE ACUERDO CON LAS PREFERENCIAS DE LOS USUARIOS

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO ESPECIALISTA EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

JUAN ENRIQUE ERAZO SÁNCHEZ

juan.erazo01@epn.edu.ec

DIRECTOR: MSc. REGINA MARITZOL TENEMAZA VERA

maritzol.tenemaza@epn.edu.ec

Quito, mayo 2019

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Juan Enrique Erazo Sánchez, bajo mi supervisión.

_____

MSc. Maritzol Tenemaza

DIRECTOR DE PROYECTO

DECLARACIÓN

Yo, Juan Enrique Erazo Sánchez, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Juan Enrique Erazo Sánchez

DEDICATORIA

A mis padres, que gracias a sus enseñanzas me han forjado como un hombre de bien.

A mi perrito Rey, por haberme acompañado durante toda mi vida académica.

A mis amigos, por todas las experiencias compartidas en los mejores años de la vida.

Y a mis profesores, que debido a sus conocimientos impartidos me han convertido en un excelente profesional.

Juan Erazo

AGRADECIMIENTO

Le agradezco cordialmente a mi directora de tesis MSc. Maritzol Tenemaza por su guía durante todo el desarrollo del proyecto; al MSc. Henry Paz por sus conocimientos brindados en la materia de Inteligencia Artificial que fueron de gran ayuda para la realización de este trabajo; y a mis compañeros de la facultad por sus consejos brindados en la recta final.

Juan Erazo

CONTENIDO

RESUMEN

El presente trabajo ofrece una solución novedosa al problema denominado *Tourist Trip Design Problem*, un problema NP-duro de optimización combinatoria que consiste en maximizar la satisfacción de un turista cuando está de visita en una ciudad. La solución propuesta es un algoritmo que trabaja sobre un conjunto de puntos de interés (POIs) obtenidos con la API de Google Maps basándose en las preferencias del turista; y utiliza las bondades de dos algoritmos de inteligencia artificial para procesar los datos. El primero, es el algoritmo de *clustering kmeans* que permite clasificar a los POIs por sectores geográficos; y el segundo, es un algoritmo genético que optimiza el tiempo invertido para conocer los sitios de cada sector. Como resultado, se obtiene una recomendación de itinerarios turísticos con tiempos optimizados que se ajustan a los gustos del turista.

Palabras clave: Algoritmos**,** Optimización combinatoria, complejidad computacional, problemas NP, puntos de interés, API, *kmeans,* algoritmos genéticos.

ABSTRACT

The present text offers a novel solution to the Tourist Trip Design Problem, a NP-hard combinatorial optimization problem that deals with of maximizing the satisfaction of a tourist when visiting a city. The proposed solution is an algorithm that uses the benefits of two artificial intelligence algorithms to process a set of points of interest (POIs) obtained with the Google Maps API, which correspond to the preferences of a tourist. The algorithm starts using the *kmeans* algorithm to classify the POIs by geographical sectors; then, the visiting time of each sector is optimized by using a genetic algorithm. The output is a recommendation of optimized tourist itineraries that adjust to the tourist's preferences.

Key words: Algorithms**,** combinatorial optimization, computational complexity, NP problems, points of interests, API, *kmeans,* genetic algorithms.

1 INTRODUCCIÓN

1.1 Planteamiento del problema

En el sector turístico, los viajeros disponen de un tiempo limitado para visitar todos los sitios turísticos que sean de su interés. Considerando que existen ciudades cosmopolitas, la organización y planificación de un itinerario de acuerdo con sus intereses podría llegar a ser una complicación y el turista se marcharía de la ciudad sin haber conocido lugares importantes. Adicionalmente, las empresas turísticas también requieren apoyo en este tipo de definiciones.

Este problema es conocido como "*Tourist Trip Design Problem*" (TTPD), cuyo objetivo consiste en maximizar la satisfacción de un turista al diseñar rutas con los puntos de interés que sean de su preferencia. Actualmente, existen diferentes propuestas para la solución de este problema de optimización combinatoria [1], en donde se utilizan metaheurísticas [2] como algoritmos de fuegos artificiales (*Fireworks Algorithm*) [3], algoritmos GRASP [4], algoritmos genéticos [5] [6], etc.

Como nueva solución, en este trabajo se propone el desarrollo de un algoritmo híbrido que fusione al algoritmo de *clustering K-means* [7] con un algoritmo genético personalizado [8]. De esta manera, se pretende mejorar la distribución del tiempo disponible que tendría el turista, puesto que *K-means* permitirá agrupar los sitios turísticos en sectores geográficos, y el algoritmo genético calcularía un itinerario óptimo para cada sector, tomando en cuenta estimaciones de tiempo y horarios de atención.

1.2 Objetivos

1.2.1 Objetivo General

Desarrollar un recomendador de itinerarios turísticos considerando los intereses de los usuarios mediante la combinación del algoritmo de clustering K-means con un algoritmo genético.

1.2.2 Objetivos Específicos

- Determinar los puntos de interés (POIs) para el usuario, considerando su posición en una determinada ciudad y sus preferencias.
- Diseñar rutas e itinerarios turísticos de acuerdo con los días de estancia del viajero.

- Optimizar los POIs para el turista, de manera que haya una mejor calidad de sitios turísticos por visitar.

1.3 Marco Teórico

1.3.1 Complejidad Algorítmica

La teoría de la complejidad computacional se enfoca en medir la cantidad de recursos computacionales requeridos para resolver una tarea. Se basa en dos parámetros: la memoria utilizada y el tiempo que tarda en ejecutarse un algoritmo. Esto permite determinar cuál es la manera más eficiente para resolver un problema [9].

El conjunto de problemas de complejidad computacional se diferencia en diferentes clases. Cada clase es definida mediante el tipo de problema (generalmente problemas de decisión), el modelo computacional y la medida de su complejidad [10]. Entre las diferentes clases se puede hallar a los problemas P, NP, NP-completos, etc.

La clase de problemas NP son aquellos donde se puede verificar en tiempo polinómico si una solución resuelve al problema, se clasifican en dos grupos [1]:

- Los problemas NP, que obtienen su solución mediante un algoritmo que se ejecuta en tiempo polinomial.
- Y los problemas NP-completos, donde se considera que no existe un algoritmo polinómico que solucione el problema, aunque, todavía no se ha logrado demostrar matemáticamente su inexistencia.

Además, existe otra clase de problemas denominados NP-duros (problema tratado en el trabajo actual) que pueden incluso ser más difíciles de resolver que los NP-completos. Los problemas NP-duros no son un subconjunto de los problemas NP ya que no existe un algoritmo en tiempo polinómico que pueda validar si una solución es correcta [1]. En la Figura 1 se representa la relación de las clases de problemas mencionados.

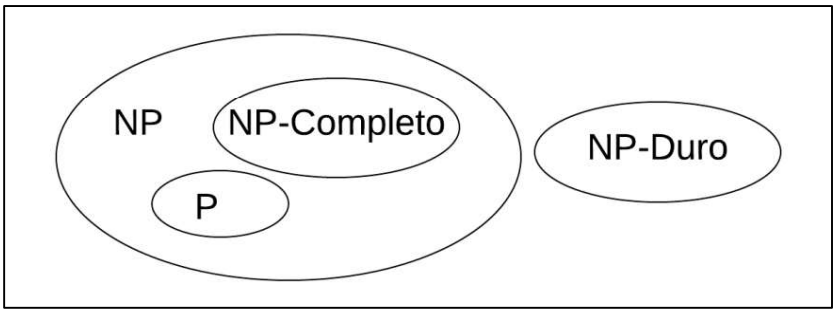

Figura 1. Relación entre los problemas P, NP, NP-completo, NP-duro

La mayoría de los problemas con un enfoque práctico y científico no tienen un algoritmo con una complejidad polinómica que lo resuelva. Dado el interés que tiene la solución de estos problemas de forma exacta, se tiene como otra opción encontrar soluciones de alta calidad en un tiempo moderado. Para estas tareas se necesita algoritmos aproximados que permitan resolver problemas de optimización [1].

1.3.2 Optimización

La optimización es una disciplina abarcada dentro de las Ciencias de la Computación, Inteligencia Artificial y la Investigación Operativa [1]. Optimización significa tratar de encontrar la mejor solución dentro de un conjunto de soluciones factibles (aquellas que satisfacen todas las restricciones del problema de optimización), la cual podría minimizar el costo de un proceso o maximizar la eficiencia de un sistema [11].

Un problema de optimización presenta varias (en realidad muchas) posibles soluciones y alguna forma clara de comparación entre ellas, de manera que éste existe si y sólo si se dispone un conjunto de soluciones candidatas diferentes que pueden ser comparadas [1]. Existen dos tipos de problemas de optimización [1]:

- Los problemas cuyo conjunto de soluciones lo conforman los números reales (optimización continua).
- Y los que su conjunto de soluciones está conformado por números enteros (optimización discreta).

La segunda categoría abarca un tipo particular de problemas denominados “problemas de optimización combinatoria” que corresponde al problema tratado en el presente proyecto.

1.3.3 Optimización Combinatoria

La optimización combinatoria (OC) es el proceso de encontrar una solución óptima dentro de un conjunto discreto y finito de posibles soluciones, el cual suele ser bastante amplio. Las soluciones están representadas mediante alguna estructura matemática y su calidad dependerá en su capacidad para minimizar o maximizar una determinada función (denominada función objetivo) cumpliendo sus restricciones establecidas [12].

La OC tiene numerosas aplicaciones en ámbitos como las ciencias, la industria, la ingeniería, la logística, etc. Algunos casos prácticos son: el enrutamiento y carga de vehículos en redes de distribución, el diseño de proteínas, la asignación de tareas a procesadores, el diseño de redes de telecomunicación, la planificación de la producción, la distribución de ambulancias en una región, etc. [12]

Formalmente, se puede definir a la OC como una tripla (S, f, Ω), en donde:

- S es el conjunto de soluciones posibles.
- f es la función objetivo, que debe ser maximizada o minimizada.
- Ω es un conjunto de restricciones que deben ser cumplidas.

Algunos ejemplos muy conocidos de problemas de optimización combinatoria son [1]:

- Problema del vendedor viajero (*Traveling Salesman Problem*): Posiblemente es el problema más conocido y estudiado dentro de la OC. Se lo describe con el siguiente enunciado: “Dado un conjunto de ciudades y las distancias de separación entre ellas ¿Cuál sería la ruta más corta posible para que un vendedor viajero visite cada ciudad exactamente una vez y termine regresando a la que partió?” (ver Figura 2)
  Es un problema de tipo NP-duro y presenta numerosas aplicaciones en diferentes áreas ya que, de éste se derivan una gran cantidad de problemas de enrutamiento, entre ellos, el problema tratado en el presente trabajo.

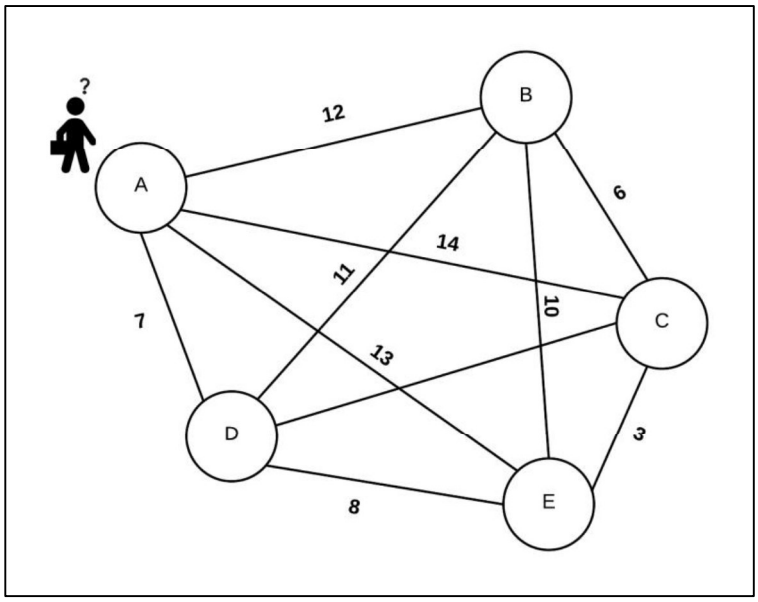

Figura 2. Problema del agente viajero

- El problema de la mochila: también conocido como *Knapsack Problem (KP),* trata de encontrar una combinación adecuada de objetos (cada uno con peso y valor determinados) para que encajen en una mochila, de manera que, se debe maximizar el valor total sin exceder el peso máximo que ésta pueda soportar.
- El problema de enrutamiento de vehículos (*Vehicle Routing Problem*) consiste en encontrar el grupo de rutas lo más cortas posible, utilizando un conjunto de carros (el más pequeño posible). Tal que, los vehículos deberían abastecer a una serie de clientes distribuidos geográficamente. Se debe tomar en cuenta que cada vehículo tiene una capacidad máxima propia, y la cantidad de producto demandada por los clientes varía.
- El problema de la ordenación lineal (*Lineal Ordering Problem*) intenta encontrar una permutación de columnas y filas dentro de una matriz, de modo que se maximice la suma de los pesos en el triángulo superior de ésta.

Cada problema de OC puede ser resuelto a través del algoritmo de búsqueda por fuerza bruta o exhaustiva, que consiste en enumerar todas las posibles soluciones candidatas y seleccionar la mejor. Sin embargo, esta solución es ineficiente ya que el espacio de soluciones crece de manera exponencial con el tamaño del problema. Es por lo que, existen algoritmos aproximados que permiten encontrar soluciones de alta calidad en un tiempo computacional breve; para ello, se utilizan las denominadas técnicas heurísticas y metaheurísticas.

1.3.4 Tourist Trip Design Problem

El TTDP (*Touris Trip Design Problem)* es el problema de OC que se plantea resolver en el presente proyecto, consiste en diseñar un enrutamiento para turistas que están interesados en visitar varios puntos de interés (POIs) como se muestra en la Figura 3. Las soluciones vendrían a ser el orden en cómo visitar los POIs respetando las restricciones del turista y los atributos de los POIs [13].

El principal objetivo del problema es seleccionar los POIs que coincidan con las preferencias de los turistas, maximizando así su satisfacción. Se debe tener en cuenta una multitud de parámetros y restricciones como las distancias entre los POIs, el tiempo de visita requerido para cada POI, los días de visita, las tarifas de entrada, las condiciones climáticas, etc. respetando el tiempo disponible que se tendría para hacer turismo en un día [13].

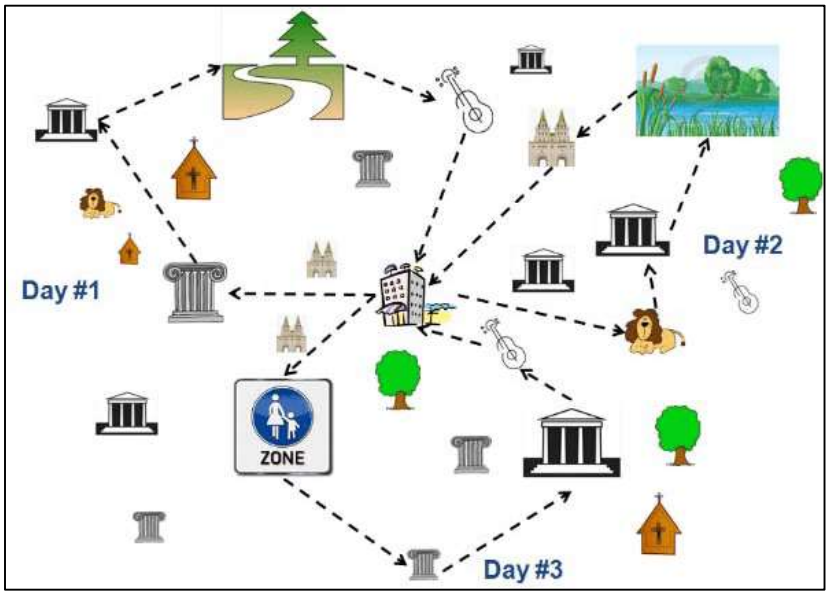

Figura 3. *Tourist Trip Design Problem* [13]

1.3.5 Heurísticas

Los problemas de OC pertenecen a la clase de complejidad NP, por lo cual, no se puede garantizar encontrar una solución adecuada en un tiempo polinómico razonable. Sin embargo, existen otras alternativas conocidas como “heurísticas” que permiten encontrar soluciones de calidad en tiempos eficientes.

“Heurística” es un término de origen griego, proviene de la palabra *heuriskein* que significa “encontrar” o “descubrir”. Por otra parte, según la historia se deriva de *eureka*, famosa exclamación atribuida a Arquímedes [2]. Ya que etimológicamente la heurística se enfoca en el estudio de la invención y el descubrimiento, se puede relacionar a la heurística con la tarea de resolver problemas inteligentemente utilizando la información que se tiene al alcance [2].

Los métodos heurísticos se pueden clasificar en [2]:

- Métodos de descomposición.
- Métodos inductivos.
- Métodos de reducción.
- Métodos constructivos.
- Métodos de búsqueda local.

La principal limitación de las heurísticas es su incapacidad para escapar de óptimos locales. Por ende, estos algoritmos no utilizan ningún mecanismo para proseguir la búsqueda de una mejor solución en caso de haber caído en un óptimo local [1].

1.3.6 Metaheurísticas

En la búsqueda de mejores soluciones y de mayor calidad, aparecen las metaheurísticas que son estrategias para diseñar y mejorar los procedimientos heurísticos con el fin de obtener un alto y mejor rendimiento [2].

El término “metaheurística” fue introducido por Fred Glover en 1986, etimológicamente deriva de la composición de las palabras “meta” y “heurística”, en donde el prefijo meta (en inglés) significa “más allá” o “en un nivel superior” [1].

Se pueden describir los siguientes tipos de metaheurísticas [14]:

- Metaheurísticas de relajación: Son procedimientos que realicen modificaciones (relajaciones) al modelo original, haciéndolo más fácil de resolver. Ejm: el método de relajación lagrangiana.
- Metaheurísticas constructivas: Se refieren a los procedimientos que tratan de obtener la solución a partir del análisis de los componentes que la conforman, utilizando un procedimiento que iterativamente agrupa elementos a una estructura

(inicialmente vacía). Ejm: la estrategia voraz o *greedy*, la metaheurística GRASP (*Greedy Randomized Adaptive Search*), etc.

- Metaheurísticas de búsqueda: Técnicas que usan movimientos o transformaciones para examinar el espacio de soluciones alternativas y aprovechar las estructuras de entornos relacionadas. Ejm: el recocido simulado (*Simulated Annealing*), la búsqueda tabú (*Tabu search*), etc.
- Metaheurístias evolutivas: se orientan a procedimientos que obtienen soluciones óptimas a través de la evolución en el espacio de búsqueda del conjunto de soluciones. Un ejemplo claro son los algoritmos genéticos, que se basan en la selección natural, donde los individuos más aptos de una población sobreviven a los cambios que se producen en su entorno, y tras su reproducción permiten crear generaciones nuevas con mejores características (ver Figura 4).

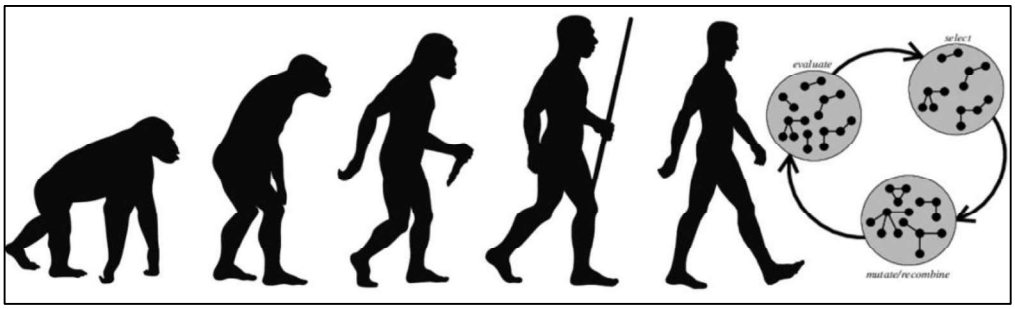

Figura 4. Metaheurísticas evolutivas [15]

Existen otras metaheurísticas de búsqueda que se inspiran en fenómenos de la naturaleza. Tal como: la optimización basada en colonias de hormigas (*Ant Colony Optimization*) que es una metaheurística relativamente nueva inspirada en el comportamiento estructurado de las colonias de hormigas para su comunicación a través de feromonas; las redes neuronales artificiales. Etc.

1.3.7 Algoritmos Genéticos

Los algoritmos genéticos (AGs) son metaheurísticas de búsqueda utilizados en problemas de optimización combinatoria y están basados en la mecánica de la selección natural y la genética. Combinan la supervivencia del más apto entre estructuras con un intercambio de información estructurado y aleatorio [16]. En el presente proyecto, se utilizaron los beneficios de los AGs para optimizar el tiempo de recorrido en los itinerarios turísticos.

El pionero de la teoría de los AGs es John Henry Holland, investigador de la universidad de Michigan, en 1975 publicó "*Adaptation in natural and artificial systems*" en donde describe cómo aplicar los principios de la evolución natural a los problemas de optimización, diseñando el primer AG. La teoría de Holland se ha hecho más robusta con el pasar de los años y hoy en día, los AGs son una herramienta muy poderosa para resolver problemas de optimización combinatoria [17].

En la Figura 5 se puede ver el diagrama de flujo de un algoritmo genético, empieza con la inicialización de una población con posibles soluciones al problema. A partir de ésta, se crearán nuevas soluciones mediante un cruce aleatorio entre los mejores elementos de la población; las nuevas soluciones serán evaluadas respecto a sus características, de modo que las más pobres serán descartadas y las más aptas seguirán durante las siguientes generaciones. Todo este proceso es iterativo hasta que cumpla un criterio de aceptación que lo finalice [18].

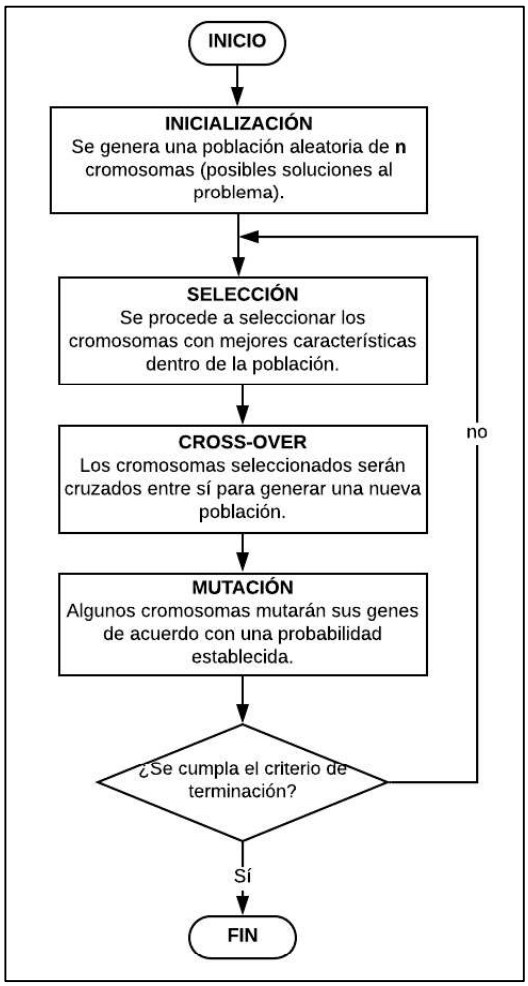

Figura 5. Algoritmo Genético Simple

1.3.7.1 Codificación de las soluciones

Un AG trabaja con un conjunto de soluciones candidatas que se ajustan a la solución del problema que se quiere resolver. Cada solución se codifica en una cadena de valores llamada **cromosoma;** la codificación juega un rol importante ya que determina las operaciones que se realizarán sobre los cromosomas [19]. Existen algunos tipos de codificación que se adecúan de acuerdo con el tipo de problema (Ver Tabla 1). En el presente proyecto, se codificaron las soluciones mediante permutaciones de números para describir el orden de visita de los diferentes POIs.

Tabla 1. Tipos de codificación para un AG [17]

Codificación	Características	Ejemplo
Binaria	- La codificación más utilizada. - Cada cromosoma es codificado mediante una cadena de bits (0s o 1s). - Cada bit representa una característica de la solución. - La longitud de la cadena depende de la precisión.	Cromosoma A: 10100101010101 Cromosoma B: 00010101010101
Octal	- La codificación se la realiza a través de cadenas de números octales (0-7).	Cromosoma A: 034671253 Cromosoma B: 102637241
Hexadecimal	- Los cromosomas son cadenas compuestas por números hexadecimales (0-9,A-F)	Cromosoma A: AF279A Cromosoma B: DB18E9
De permutación	- Cada cromosoma representa una cadena secuencial de números. - Es utilizada para problemas de ordenación - En algunos casos se deben realizar correcciones después de alguna operación genética.	Cromosoma A: 1726354 Cromosoma B: 7253416

1.3.7.2 Función Fitness

La función fitness es un tipo particular de función objetivo que permite evaluar la calidad de las posibles soluciones (cromosomas) de un AG. A parte de indicar qué tan buena es una solución, también puede mostrar qué tan cerca está un cromosoma de ser óptimo. Una función fitness ideal se adapta con el objetivo de optimización del algoritmo y gracias a ella, se puede realizar el proceso de selección [20]. La función fitness fue la clave en el desarrollo del proyecto ya que permitió evaluar los itinerarios durante la ejecución del AG.

1.3.7.3 Selección

La fase de selección será la responsable de elegir a los individuos para la reproducción basándose en sus características. Por lo que, los individuos que sean más aptos tendrán más oportunidades de ser elegidos. Sin embargo, no se debe descartar del todo a los individuos menos aptos ya que se perdería variabilidad genética [16]. Existen algunos algoritmos que son utilizados para este fin y entre ellos se pueden mencionar:

Selección por ruleta

La selección por ruleta es una de las técnicas más tradicionales de selección en los AG y la que se utilizó en el actual proyecto. Consiste en utilizar un mecanismo de ruleta para seleccionar probabilísticamente a los individuos basándose en su rendimiento. Se empieza sumando el valor fitness de todos los individuos de la población. Luego, cada individuo es mapeado en intervalos contiguos dentro de un determinado rango [0, suma total], por lo que el tamaño de cada intervalo corresponde al valor fitness del individuo. Finalmente, se genera un número aleatorio entre el 0 y la suma total de los fitness, y se selecciona al individuo que se encuentra dentro de ese intervalo, los individuos con un fitness más alto tendrán más oportunidades de ser seleccionados [21]. En la Figura 6 se puede visualizar cómo la probabilidad de ser seleccionado varía en cada individuo de acuerdo con su fitness.

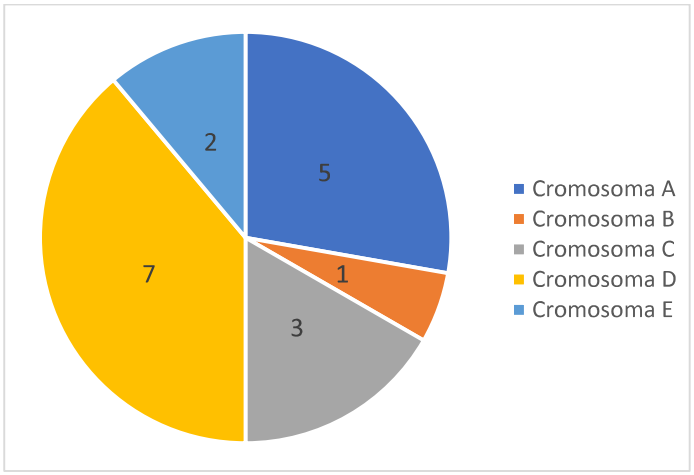

Figura 6. Selección por ruleta [23]

Selección por ranking

La selección por ruleta presenta problemas de convergencia cuando el fitness de algún cromosoma difiere mucho con respecto de los demás. Si un cromosoma ocupa el 90% de la ruleta, los demás se encontrarían perjudicados en la selección (ver Figura 7). Como solución, aparece el algoritmo de selección por ranking que empieza ordenando la población de cromosomas de acuerdo con su valor fitness original; luego, establece un nuevo fitness conforme a su posición; y, por último, ejecuta el mismo procedimiento de selección por ruleta. De este modo, el cromosoma con el peor fitness tendrá un valor de 1 y el cromosoma con el mejor fitness tendrá un valor de N, que corresponde al tamaño de la población [17] (ver Figura 8).

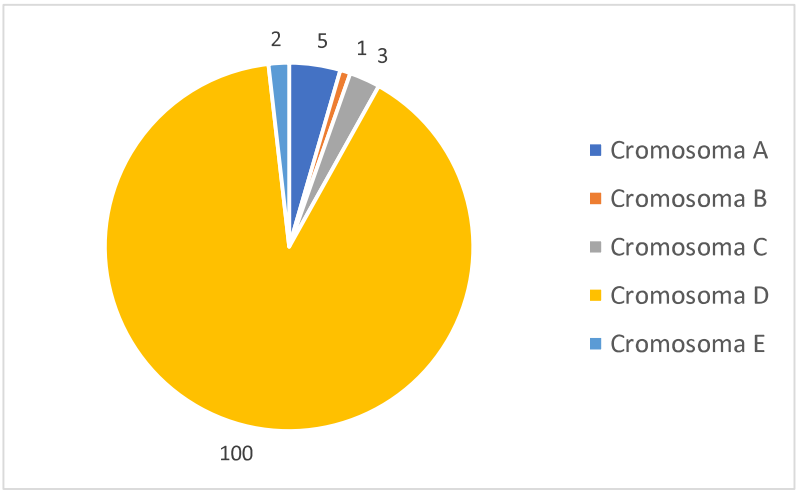

Figura 7. Selección por ranking (antes de ordenar la población de cromosomas) [23].

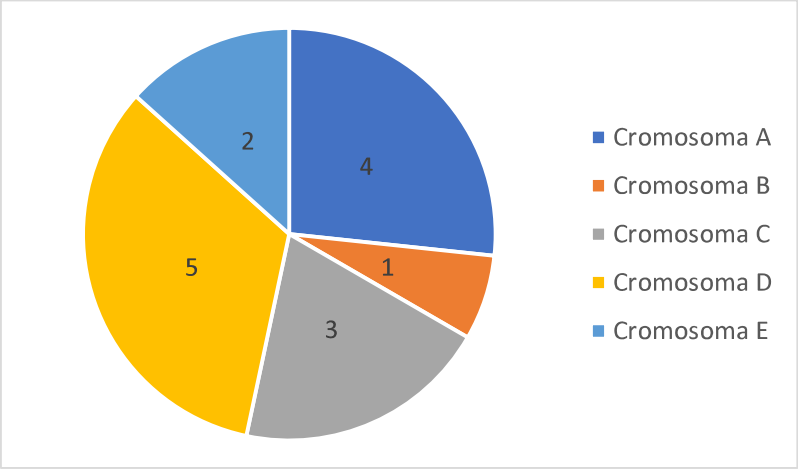

Figura 8. Selección por ranking (después de haber ordenado la población de cromosomas) [23].

Selección por torneo

La selección por torneo ofrece una estrategia para afinar el rendimiento del AG mediante una serie de torneos entre grupos de cromosomas o individuos (generalmente 2), donde se selecciona al que tenga el mejor fitness [17]. Existen dos tipos: el torneo determinístico, que selecciona al individuo más apto dentro de un conjunto aleatorio de individuos; y el torneo probabilístico, que utiliza un número aleatorio para escoger al individuo, que podría ser el más apto o el menos apto [16].

1.3.7.4 Crossover

Después de haber seleccionado a los individuos, se sigue con el proceso de cruce entre ellos. El c*rossover* o cruce es un operador que permite realizar la recombinación entre dos o más cromosomas para producir descendencias que formarán parte de una nueva generación. La idea fundamental del *crossover* es que los nuevos individuos heredarán las mejores características de sus padres y representarían una mejor solución [16]. Existen diferentes tipos de cruce, de los cuales se destacan los siguientes:

Cruce de un punto

Es el algoritmo más básico de *crossover,* consiste en seleccionar un punto aleatorio de corte entre los padres y generar las descendencias al intercambiar las secciones cortadas entre ellos [16] (ver Figura 9).

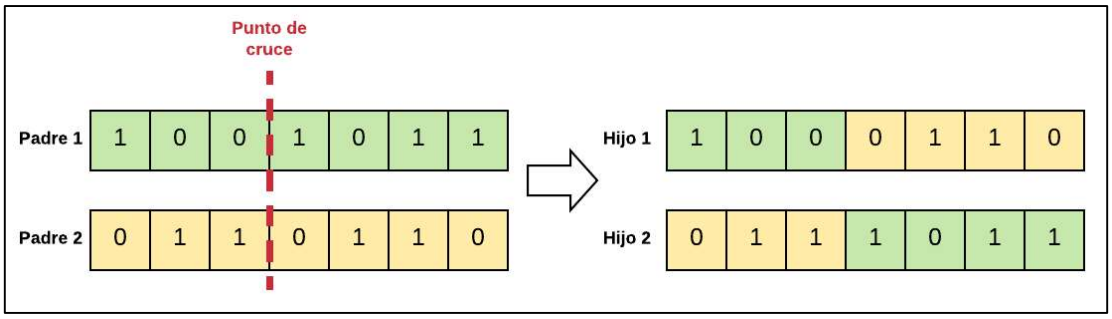

Figura 9. Cruce de un punto

Cruce de dos puntos

Varios algoritmos aparecieron a partir de la técnica de cruce por un punto, entre ellos se encuentra el cruce de dos puntos, que trata de fijar dos puntos de cruce aleatorios en lugar de uno (ver Figura 10). Cabe recalcar que al agregar más puntos de corte se reduce la eficiencia del algoritmo, pero se puede explorar más posibilidades dentro del conjunto de soluciones [17].

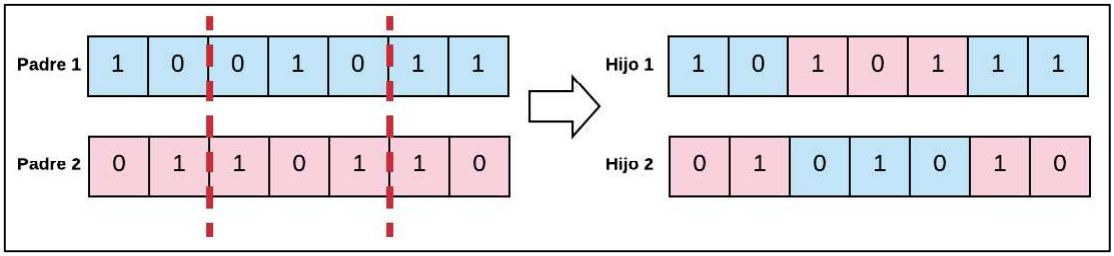

Figura 10. Cruce de dos puntos

Cruce uniforme

En el cruce uniforme, cada gen del nuevo cromosoma tiene la misma probabilidad de pertenecer a alguno de los dos padres. Para ello, se utiliza una máscara de valores aleatorios binarios que tiene la misma longitud de los cromosomas padres [16]. En la Figura 11 se puede visualizar que cuando una posición de la máscara tiene el valor de 1, se toma el gen del primer padre; caso contrario, si es 0, se asigna el gen del segundo padre.

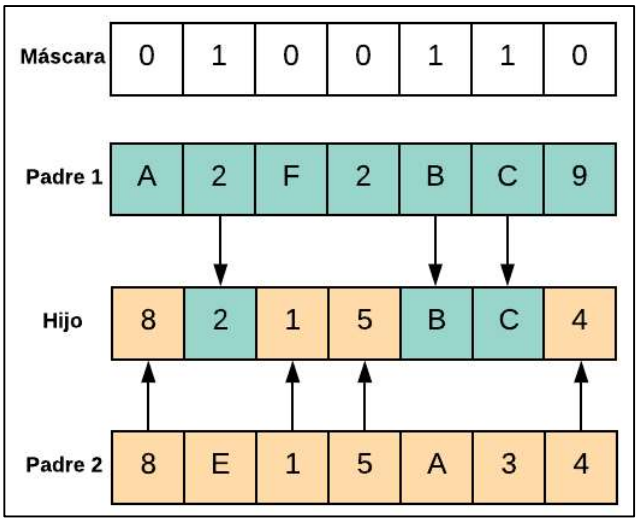

Figura 11. Cruce uniforme

Partially-mapped crossover (PMX)

En el caso de las soluciones codificadas por permutación, no se puede utilizar ninguna de las técnicas mencionadas anteriormente ya que se alteraría la secuencia de números y se obtendrían genes repetidos. Como alternativa, aparecen otros algoritmos de *crossover* que trabajan con este tipo de codificaciones, entre ellos, se puede nombrar al *partially-mapped crossover (PMX)* que fue utilizado en el presente proyecto.

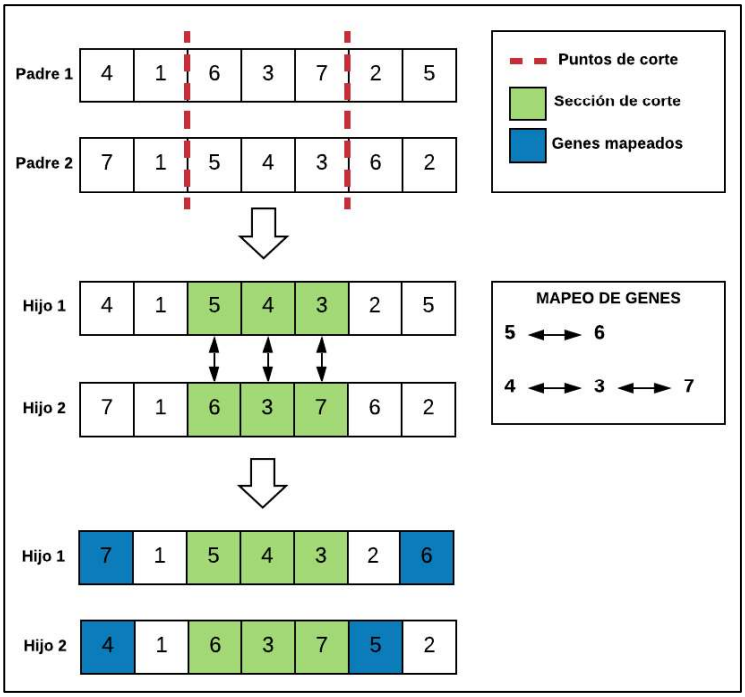

Figura 12. Partially-mapped crossover

El PMX empieza seleccionando dos puntos de corte aleatorios y ejecuta el mismo procedimiento de cruce por dos puntos. Posteriormente, en las nuevas soluciones se realiza un mapeo de genes fuera de la sección de corte con el fin de eliminar los duplicados (Ver Figura 12). El mapeo corresponde a un reemplazo inverso de los genes que fueron seleccionados en el área de corte [8].

1.3.7.5 Mutación

Después del *crossover,* los cromosomas son sujetos a un proceso de mutación, donde se modifica su código genético de manera probabilística. La mutación juega un rol importante ya que ayuda a mantener la diversidad genética en la población y permite explorar más opciones dentro del espacio de soluciones [17]. Existen algunos tipos de mutación que varían de acuerdo con la codificación de las soluciones; por ejemplo, para las codificaciones binarias, se puede negar un gen (cambiar su valor de 0 a 1 o viceversa) de acuerdo con una taza de mutación; en cambio, para las codificaciones de permutación (utilizadas en el proyecto), se puede seleccionar genes aleatorios dentro del cromosoma e intercambiar sus posiciones [16].

1.3.7.6 Parámetros de un AG

Para que un AG tenga éxito en la optimización, se deben escoger los parámetros correctos. Algunas taxonomías se diferencian por parámetros exógenos y endógenos. Los parámetros exógenos son parámetros genéricos que definen las propiedades globales de un AG como el tamaño de la población o probabilidad de *crossover.* Mientras que, los parámetros endógenos se definen propiedades más específicas que afectan a la codificación de las soluciones [22]. Para el desarrollo del AG, se utilizaron tanto parámetros endógenos como exógenos. A continuación, se listan unos ejemplos de parámetros exógenos [23]:

Probabilidad de *crossover*

Este parámetro significa con qué frecuencia se realizará el *crossover* dentro de una población. Si la probabilidad valiese 100%, se seleccionarían a todos los elementos de la población sin importar sus características para generar las nuevas descendencias. En cambio, si fuera 0%, las nuevas generaciones serían una copia exacta de sus padres.

Probabilidad de mutación

La probabilidad de mutación representa la frecuencia con la que podrían mutarse los genes de un cromosoma. Si la probabilidad equivale a 100%, todo el código genético cambiará; pero si es 0%, seguirá siendo igual.

Tamaño de la población

Dice cuántos cromosomas en total hay dentro de una población. Si hay muy pocos cromosomas, el AG explorará sobre una pequeña parte del espacio de soluciones. Por otro lado, si hay demasiados cromosomas, el AG se ralentiza. Algunas investigaciones demuestran que después de algún límite no es factible aumentar el tamaño de la población, ya que no hace que la solución del problema sea más rápida [23].

1.3.8 K-Means

En el proyecto actual se empleó k-means para agrupar los diferentes POIs en sectores geográficos. K-means es un algoritmo de aprendizaje no supervisado utilizado para resolver problemas de agrupamiento o *clustering*. El objetivo de un problema de agrupamiento es encontrar $\boldsymbol{k}$ $(1 \leq k \leq n)$ grupos o clústeres $c = \{c_1, c_2, \dots, c_n\}$ dentro de un conjunto $\boldsymbol{x}$ de $\boldsymbol{n}$ elementos $x = \{x_1, x_2, \dots, x_n\}$, donde la similitud entre los elementos de cada clúster es maximizada [7].

K-means utiliza un procedimiento iterativo donde cada iteración intenta minimizar la suma de las distancias euclidianas entre los elementos de cada clúster con el fin de maximizar la similitud entre ellos [7] (ver ecuación 1). Las distancias se calculan con respecto a un eje o centroide $\boldsymbol{\mu}$ fijado dentro de cada clúster $\boldsymbol{c}$ en cada iteración.

$$\sum_{j=1}^{k} \sum_{x_i \in c_j} ||x_i - \mu_j||^2$$

Ecuación 1. Suma de la distancias euclidianas entre los elementos de cada clúster

Restricciones

Se pueden agregar restricciones dentro del algoritmo k-means, tales como fijar el tamaño de cada grupo o clúster [7] (ver Figura 13). En el presente trabajo se utilizó esta restricción

con el fin de equiparar el número de sitios por cada sector geográfico y no tener inconsistencias al momento de generar los itinerarios turísticos.

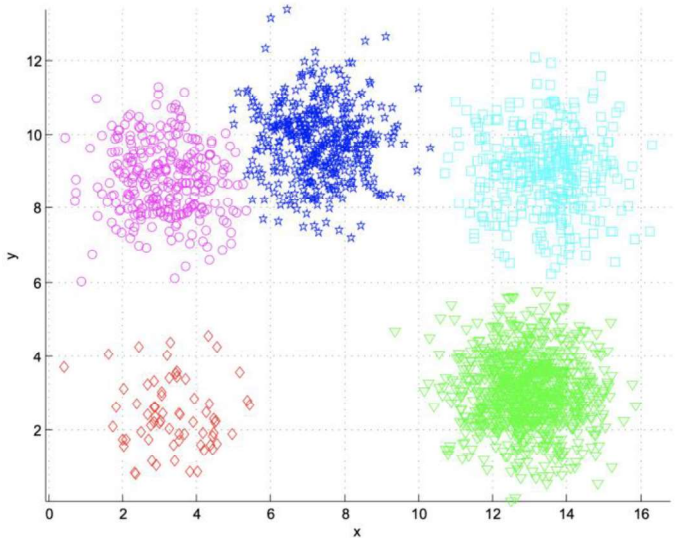

Figura 13. K-emanes con restricciones de clústeres [7]

1.4 Herramientas, frameworks y librerías de desarrollo

Express.js

Express.js o Express es un *framework* minimalista y flexible para desarrollar aplicaciones web de Node.js, proporciona un conjunto sólido de características para crear una API de manera rápida y fácil [24]. Este *framework* fue empleado para construir el servicio web que ejecuta al algoritmo híbrido.

Places API de Google Maps

Es un servicio que devuelve información sobre lugares utilizando solicitudes HTTP. Los lugares se definen dentro de esta API como establecimientos, ubicaciones geográficas o puntos de interés prominentes (POIs) [25]. En el desarrollo del proyecto se utilizaron las siguientes solicitudes [25]:

- *Place Search*: devuelve una lista de lugares según una ubicación en específico o una cadena de búsqueda.

- *Place Details:* devuelve información más detallada sobre un lugar específico, incluyendo las opiniones de los usuarios.

Directions API de Google Maps

Es un servicio que calcula las direcciones, rutas, tiempos estimados, etc. entre dos ubicaciones; permite especificar varios modos de transporte como conducir un carro, caminar, tomar transporte público o andar en bicicleta [26]. Mediante este servicio, se pudo obtener los tiempos de traslado entre los diferentes sitios turísticos, incluyendo sus rutas.

React

Se utilizó React para desarrollar interfaces del usuario final. React es una librería *Javascript* creada por Facebook para abordar algunos de los desafíos asociados con los sitios web a gran escala y basados en datos [27].

Github

Github es una plataforma y servicio basado en la nube que ayuda a los desarrolladores almacenar y mantener el código de sus proyectos de software utilizando *git,* un sistema de control de versiones de código abierto creado por Linus Torvalds en 2005 [28]. Durante todo el progreso del proyecto, se utilizó Github para versionar y almacenar el código.

Webstorm

Webstorm es un IDE (*Integrated Development Environment*) robusto de la compañía Jetbrains; es utilizado para el desarrollo de modernos ecosistemas en *Javascript* y se acopla a los frameworks y librerías más actuales para proyectos web, móviles y de escritorio [29]. Este IDE fue una herramienta muy útil en el desarrollo del proyecto ya que se acopló fácilmente con todos los frameworks y librerías *javascript* utilizadas.

Postman

Postman es un cliente HTTP disponible para cualquier sistema operativo que permite analizar cualquier API RESTFul de una manera cómoda y sencilla. Gracias a Postman, se pudo consumir los diferentes servicios web, tanto de Google como del proyecto.

Moment.js

Es una librería ligera de *Javascript* sumamente poderosa ya que permite analizar, validar, manipular e imprimir fechas en cualquier formato sin importar la zona horaria [30]. Mediante esta librería, se pudo trabajar de una manera más sencilla la gestión de los tiempos a la hora de optimizar los itinerarios turísticos.

Mocha

Mocha es un *framework* robusto de JavaScript utilizado para pruebas unitarias, se ejecuta en Node.js y en el navegador. Las pruebas de Mocha se ejecutan en serie, lo que permite crear informes flexibles y precisos, además genera excepciones no detectadas en casos de prueba correctos [31]. Este *framework* fue empleado para armar todo el conjunto de pruebas unitarias del proyecto.

Chai

Chai es una librería de aserciones de tipo TDD (*Test-Driven Development*) o BDD (*Behavior-Driven Development*) para *Node.js* que se puede acoplar perfectamente con cualquier *framework* de *javascript* para pruebas [32]. Esta librería fue utilizada para realizar todas las validaciones de la funcionalidad del código del proyecto.

1.5 Scrum

Se utilizó el marco de trabajo Scrum ya que a través de un enfoque ágil se puede cumplir con los objetivos planteados dentro del tiempo establecido asimilando cualquier cambio de requerimiento.

Scrum es un marco de trabajo para el desarrollo de productos y servicios innovadores [33]. En primer lugar, se empieza con la creación de la pila de producto (*Product Backlog*), que es una lista priorizada de las características y funcionalidades del producto a desarrollar; el trabajo es realizado por un equipo (cuyos integrantes tienen roles bien definidos) durante intervalos de tiempo denominados *Sprints.* Además, durante el desarrollo se realizan diferentes tipos de reuniones para tener una comunicación y retroalimentación eficiente entre los integrantes del equipo (en la Figura 14 se puede observar una abstracción de *Scrum*).

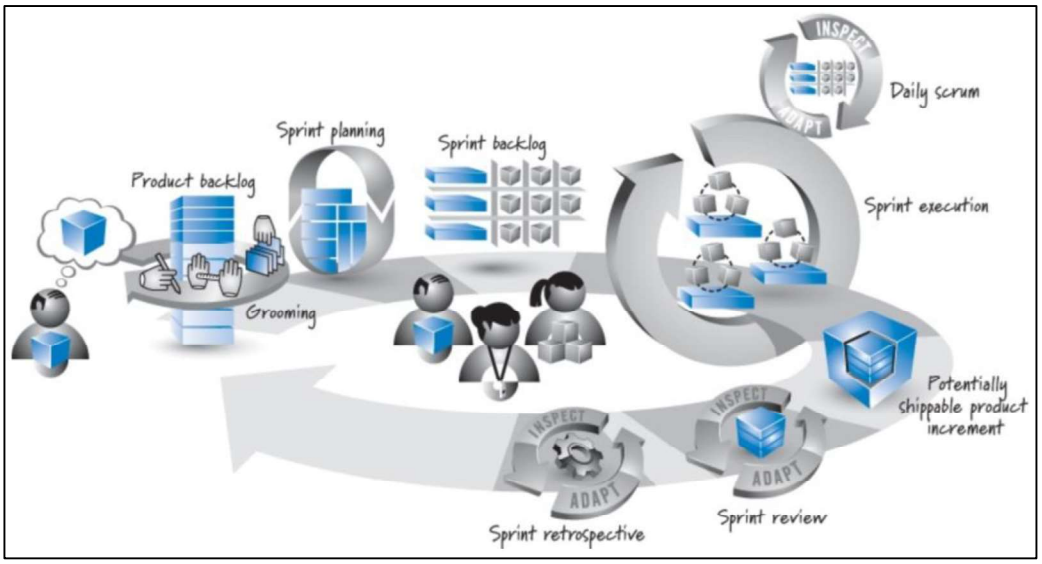

Figura 14. El marco de trabajo Scrum [33]

Origen

Las raíces de Scrum se pueden encontrar en un artículo publicado en 1986 por Takeuchi y Nonaka titulado "*The New New Product Development Game*", en el cual se describe cómo empresas de gran talla como Honda, Canon y Fuji-Xerox producen excelentes resultados utilizando un desarrollo escalable y con un enfoque basado en equipos [33].

En 1993, Jeff Sutherland y su equipo en *Easel Corporation* crearon el proceso Scrum para utilizarlo en el desarrollo de software tras combinar los conceptos del artículo de Takeuchi y Nonaka, con otros conceptos como el desarrollo orientado a objetos, control empírico de procesos, desarrollo incremental e iterativo, proceso de software y búsqueda de la productividad. En 1995, Ken Schwaber publicó el primer paper de Scrum en la OOPSLA de 1995, y desde entonces, Sutherland y Schwaber han producido varias publicaciones de Scrum como *Agile Software Development with Scrum, Agile Project Management with Scrum* y *The Scrum Guide* [33].

1.5.1 Equipo Scrum (*Scrum Team*)

Un equipo Scrum es auto organizado, multifuncional y está diseñado para optimizar la productividad, creatividad y flexibilidad [34]. Cada participante tiene un rol bien definido dentro del equipo, los cuales son descritos a continuación:

- El dueño de producto (*Product Owner*): Es la autoridad responsable de gestionar el *Product Backlog* mediante la priorización de los elementos que la conforman y la

optimización del trabajo que el equipo de desarrollo realiza. Además, tiene que asegurarse que la lista de producto sea clara y transparente para todos los integrantes del equipo [34].

- El *Scrum Master:* es el lider que está al servicio del equipo, responsable de ayudar a entender los valores, principios y prácticas de Scrum. También, ayuda al equipo a resolver cualquier inconveniente que se presente durante el desarrollo y gestiona cualquier impedimento externo que se presente [33].
- El equipo de desarrollo (*Develop*ment Team): es el grupo de profesionales auto-organizados encargados de diseñar, desarrollar y realizar pruebas del producto deseado [33]; durante cada *sprint,* el equipo transformará los elementos del *product backlog* en incrementos de funcionalidad potencialmente desplegables [34].

1.5.2 Artefactos de Scrum

Los artefactos son herramientas que permiten proporcionar transparencia y oportunidades para la inspección y adaptación. En Scrum, se pueden distinguir los siguientes [34]:

- Lista de Producto (*Producto Backlog*): Es una lista priorizada de todas los requisitos, características, funcionalidades, mejoras y correcciones que se efectuarán sobre el producto. Ésta evoluciona con el producto y el entorno en que se desenvuelve, es dinámica y cambia constantemente para acoplarse a las necesidades del producto, haciéndolo más competitivo y útil.
- Lista de pendientes del *Sprint (Sprint Backlog):* Es el conjunto de elementos seleccionados del *Product Backlog* que se realizarán durante el *sprint.* Además, incluye un plan para alcanzar el objetivo del *sprint* y entregar el incremento del producto. Por medio de esta lista, se puede visualizar en tiempo real todo el progreso y trabajo realizado por el equipo de desarrollo, los cuales son los responsables de modificarla y ajustarla en el transcurso del *sprint.*
- Incremento: Es el resultado obtenido al final de cada *sprint* y el valor de los incrementos de todos los *sprints* anteriores; cada incremento debe estar en condiciones de ser utilizado y cumplir con la definición de “terminado”.

1.5.3 Eventos de Scrum

En Scrum, los eventos son bloques de tiempo con una duración determinada y se diseñaron para habilitar los pilares vitales de transparencia e inspección:

- El *Sprint*: todo el desarrollo del proyecto se lo realiza a través de iteraciones o ciclos de hasta 1 mes de duración denominados *sprints* [33]. De tal forma que, cada *sprint* genera valor al cliente a través de incrementos (del producto "final") que son utilizables y potencialmente desplegables [34].

  Durante el transcurso de cada *Sprint* no se permite ningún cambio en los requerimientos ni en las tareas asignadas de manera que afecten al *Sprint Goal.* Del mismo modo, se respeta las fechas establecidas en el calendario, al finalizar un *sprint* inmediatamente comienza el siguiente.
- Planificación del Sprint (*Sprint Planning*): Es la reunión que se realiza por todo el equipo Scrum para la planificación del trabajo que se realizará durante el *Sprint.* Aquí se define el objetivo del Sprint (*Sprint Goal*), que corresponde a la meta que se quiere alcanzar mediante la implementación de la lista de producto y proporciona una guía al equipo de desarrollo sobre lo que se está construyendo en el incremento [34].
- Scrum Diario (*Daily Scrum*): Es una reunión diaria de 15 minutos llevada a cabo por el equipo de desarrollo, en donde cada miembro describe el trabajo que realizó desde la anterior reunión, lo que realizará hasta la próxima reunión y los inconvenientes que hayan surgido. Por ende, el Scrum diario permite mejorar la comunicación, identificar impedimentos relativos al desarrollo, promover la toma rápida de decisiones y mejorar el nivel de conocimiento en el equipo de desarrollo [34].
- Revisión del Sprint (*Sprint Review*): Es una actividad realizada al final de cada *sprint* para inspeccionar el incremento obtenido y adaptar la lista de producto si fuese necesario [34]. En este evento participa el equipo Scrum, clientes, *stakeholders* y miembros interesados de otros equipos; todos se encargan de revisar las nuevas características desarrolladas en un contexto global, de manera que puedan tener una visión clara sobre el progreso del trabajo y llegar a un acuerdo sobre las próximas funcionalidades que aporten más valor al negocio [33].
- Retrospectiva de Sprint (*Sprint Retrospective*): Es la reunión que se efectúa después del *Sprint Review* y antes del próximo *Sprint Planning.* Por medio de ésta, el equipo Scrum se realiza una auto inspección para crear un plan de mejoras que serán abordadas en el siguiente *sprint* [34].

1.5.4 Sprint 0

Antes de empezar un proyecto *Scrum* es necesario conocer el alcance del proyecto, diseñar de manera generalizada cualquier infraestructura, definir los roles del equipo *Scrum,* crear un *Product Backlog* del cual partir, etc. Para ello, ha aparecido un evento extra denominado *"sprint 0"* (que no consta en ninguna documentación oficial de *Scrum*), en el cual se realizan todas esas actividades, sin embargo, no debería ser llamado "*sprint"* porque no cumple con los criterios que caracterizan a un *sprint* tales como generar un incremento, ejecutar reuniones diarias, de revisión, de retrospectiva, etc. [35]

2 METODOLOGÍA

2.1 Sprint 0

Este "*sprint* 0" tuvo una duración aproximada de 1 mes. Aquí se recopiló toda la información necesaria antes de empezar el proyecto, se diseñó su arquitectura global, se definieron los roles del equipo *Scrum,* las historias épicas, el *Product Backlog* y finalmente el número de *sprints* con su duración.

2.1.1 Arquitectura

El algoritmo híbrido se lo representó en el diagrama de flujo de la Figura 15. La lógica comienza con la obtención de los puntos de interés de acuerdo con las preferencias del turista; luego, de acuerdo con los días de visita, se clasifica geográficamente en clústeres (un clúster por día) utilizando el algoritmo *kmeans;* finalmente, se calculan los itinerarios turísticos al optimizar cada clúster con el AG.

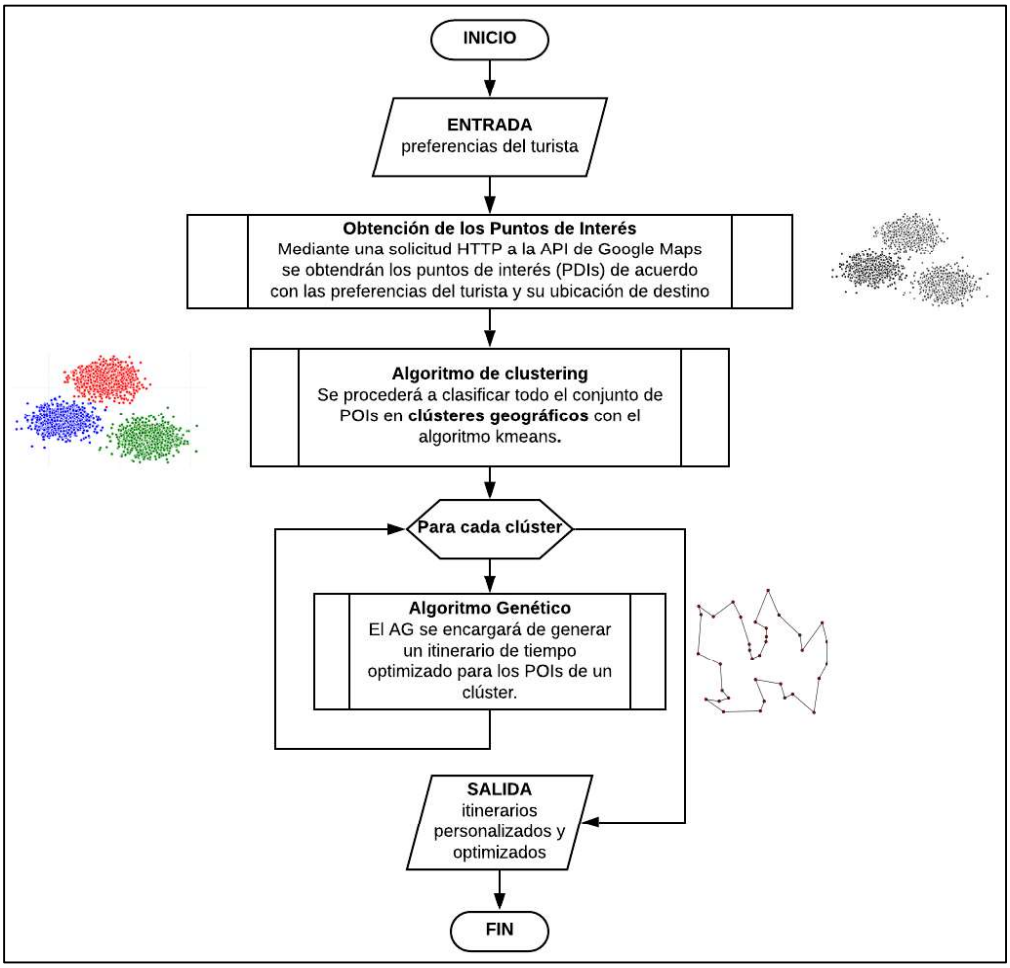

Figura 15. Diagrama de flujo del algoritmo híbrido

La Figura 16 define la infraestructura global del proyecto mediante una arquitectura Modelo-vista-controlador (MVC). El modelo corresponde a la API de Google Maps, donde se obtiene toda la información de los sitios turísticos; el controlador es un servidor que procesa las solicitudes HTTP para ejecutar el algoritmo híbrido; y la vista viene a ser la aplicación final que despliega los resultados, ya sea mediante un navegador web o un cliente REST.

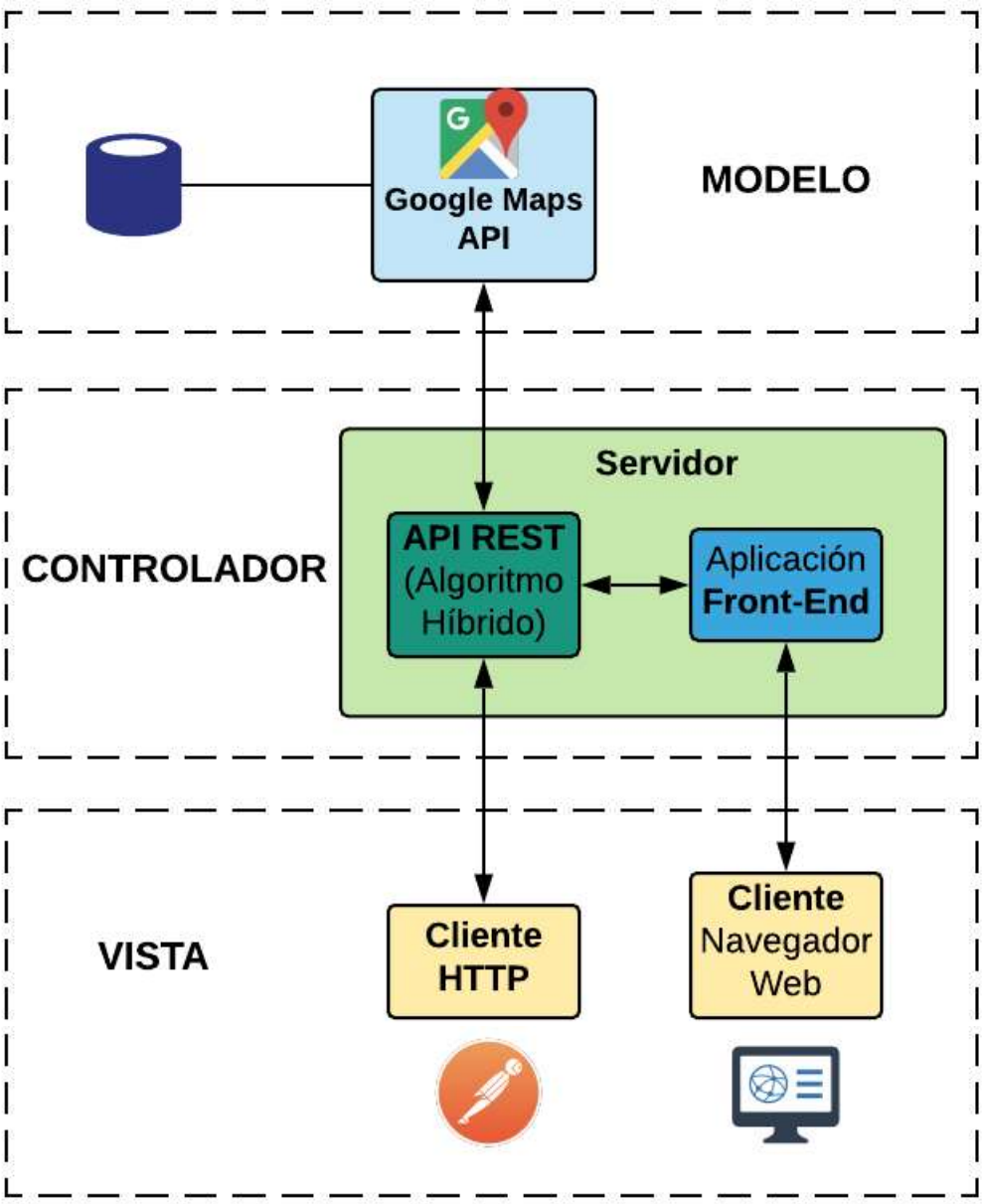

Figura 16. Arquitectura MVC del proyecto de software

2.1.2 Definición de roles

El presente proyecto sólo está conformado de 2 integrantes, por lo que un miembro del Equipo *Scrum* deberá cumplir con dos roles a la vez (ver Tabla 2). A pesar de que el equipo conste con un número pequeño de integrantes, se ha cumplido con los objetivos dentro del tiempo establecido.

Tabla 2. Roles del equipo Scrum en el proyecto

Equipo Scrum	
Rol	**Responsable**
Product Owner	Maritzol Tenemaza
Scrum Master	Maritzol Tenemaza
Development Team	Juan Erazo

2.1.3 Historias Épicas

En la Tabla 3 se puede apreciar las historias épicas identificadas para el desarrollo de todo el proyecto. El detalle de cada historia se puede visualizar en el Anexo I.

Tabla 3. Historias épicas

CÓDIGO	**TÍTULO**	**PRIORIDAD**
AH1	Integración del algoritmo de clustering con la API de Google Maps	Alta
AH2	Desarrollo del algoritmo genético personalizado.	Alta
AH3	Desarrollo de la interfaz para el usuario final	Media

2.1.4 Product Backlog

En la Tabla 4 se puede apreciar el primer *product backlog* creado en el sprint 0.

Tabla 4. *Product Backlog* inicial

PRODUCT BACKLOG				
HISTORIA ÉPICA	**HISTORIA DE USUARIO**			
	ID	**Título**	**Estimación (días)**	**Prioridad**
AH1	AH1-01	Extracción de los puntos de interés del turista.	5	Alta
	AH1-02	Agrupación geográfica de los POIs.	9	Alta
AH2	AH2-01	Generación de itinerarios turísticos optimizados.	14	Alta
AH3	AH3-01	Desarrollo del formulario del usuario final.	8	Media
	AH3-02	Desarrollo de la interfaz de despliegue de resultados.	6	Media

2.1.5 Definición de s*prints*

Tras establecer un total de 5 historias de usuario y sus respectivas estimaciones de tiempo, se tiene aproximadamente 1 mes y medio para la realización del proyecto. Por tal razón, se decidió desarrollarlo en un total de 3 *sprints,* donde cada *sprint* tendrá una duración de 2 semanas (14 días).

2.2 Sprint 1

2.2.1 Sprint Planning

Objetivo del Sprint

Construir un servicio RESTful que devuelva los puntos de interés de un turista agrupados por sectores geográficos, el número de sectores dependerá de sus días de visita.

Historias de Usuario

En la Tabla 5 se encuentran listadas las historias de usuario que fueron tomadas para el primer sprint con su respectiva estimación. En el Anexo II, se encuentran desglosadas con más detalle.

Tabla 5. Historias de usuario escogidas para el Sprint 1

ID	Título	Estimación (días)	Estado
AH1-01	Extracción de los puntos de interés del turista.	5	Por implementar
AH1-02	Agrupación geográfica de los POIs.	9	Por implementar

Sprint Backlog

Tabla 6. Sprint backlog del Sprint 1

SPRINT BACKLOG	
HISTORIA DE USUARIO	**TAREAS**
AH1-01	Crear el repositorio en Github.
	Inicializar el proyecto de NodeJs con el gestor npm.
	Instalar el *framework* Express.
	Instalar del ambiente para ejecutar las pruebas unitarias.
	Levantar el servidor y verificar que escuche solicitudes http.
	Crear una función que escuche solicitudes HTTP POST a través del *endpoint* /**ttdp** y que retorne una respuesta de tipo json.
	Diseñar el cuerpo de petición que se recibirá en la solicitud del cliente para aceptar sus preferencias y ubicación.
	Validar que el cuerpo de petición coincida con el esquema diseñado.
	Obtención de las credenciales de desarrollador para utilizar los servicios de Google Maps.
	Consumir los servicios de la API Places Search de Google Maps para obtener los puntos turísticos alrededor de la ubicación proporcionada de acuerdo con las preferencias de usuario.
	Responder los POIS extraídos de Google Maps.
AH1-02	Instalar el paquete *kmeans-same-size,* el cual ejecutará el algoritmo de clustering.
	Realizar pruebas unitarias para validar la funcionalidad del algoritmo externo.
	Modificar el esquema del cuerpo de petición para que también acepte el número de días de visita del turista.
	Clasificar los POIS obtenidos de Google Maps en **N** grupos (N es el número de días esperados de visita) utilizando el algoritmo de *clustering* kmeans.
	Retornar los clústeres calculados (cada clúster corresponderá a un sector geográfico).

2.2.2 Ejecución del Sprint

Durante la ejecución del Sprint se puede mencionar los siguientes detalles importantes:

Diagrama de secuencia

En la Figura 17 se puede visualizar el diagrama de secuencia para el servicio RESTful construido. En el cuerpo de cada solicitud HTTP se detalla las preferencias del turista, la ubicación de su destino y el número de días de visita. A partir de estos datos, se consume el *endpoint /PlaceSearch* de la *Places API* de Google Maps para obtener los puntos de interés de acuerdo con la ubicación y preferencias proporcionadas. A continuación, se ejecuta el algoritmo de *clustering kmeans* sobre el resultado obtenido para clasificar a los POIS por agrupaciones geográficas, el número de agrupaciones depende del número total de días de visita proporcionado inicialmente. Finalmente, el servicio devuelve lo obtenido a través de una respuesta en formato json siempre y cuando el cuerpo de petición contenga los parámetros requeridos; caso contrario, se retornará un error HTTP 400.

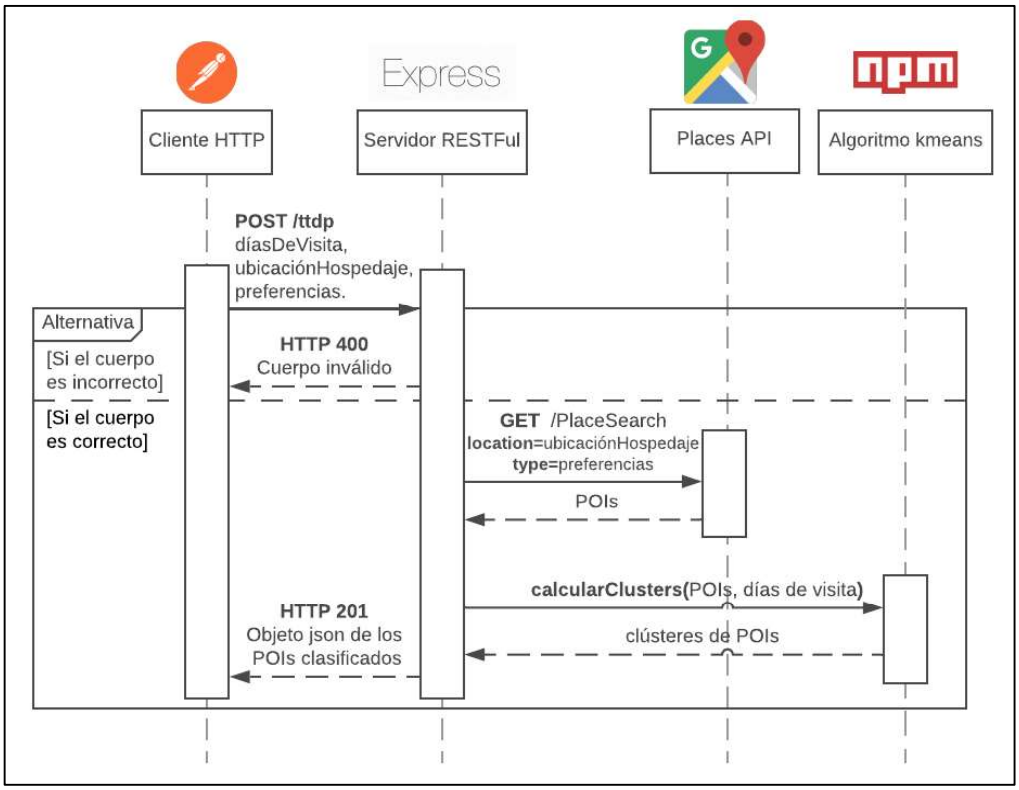

Figura 17. Diagrama de secuencia de la integración de la API de Google con el algoritmo de *clustering*

Google Maps API

Para obtener los puntos de interés con la API Places de Google Maps, se definieron los parámetros de acuerdo con los requeridos por el *endpoint* de *Place Search* [36]. Los tipos de lugares turísticos que ofrece Google [37] en su API son los siguientes:

- Museos
- Acuarios
- Exposiciones de arte
- Parques de diversiones
- Cementerios
- Iglesias
- Palacios presidenciales
- Templos hindús
- Mezquitas
- Parques
- Estadios
- Zoológicos

Algoritmo de clustering

Para la clasificación geográfica de los lugares, se utilizó el algoritmo del paquete *k-means-same-size* del repositorio *npm;* tras varias pruebas se comprobó que el algoritmo se acopla perfectamente con lo esperado, ya que se puede parametrizar el número de clústeres que se calcularán. De esa manera, cada clúster contendrá un conjunto de lugares candidatos a visitar dentro de un sector geográfico. El número de clústeres depende del número total de días de visita del turista.

Validación del cuerpo de petición

Para evitar futuros errores en la ejecución del algoritmo, se validó el cuerpo de petición que se recibe en el *endpoint* expuesto. Para ello, se utilizaron 2 paquetes de npm: *express-valitation* y *joi.* De acuerdo con el tipo de validaciones ofrecidas por el paquete *joi* [38], se diseñó el esquema que se muestra en la Figura 18.

```
body: {
    totalDays: joi.number().integer().positive().min(1).required(),
    location: joi.object().keys({
        lat: joi.number().required(),
        lng: joi.number().required(),
    }).required(),
    categories: joi.array().items(joi.string()).required()
}
```

Figura 18. Esquema inicial del cuerpo de petición de acuerdo con las validaciones de joi [38]

De esta manera se puede verificar que:

- *totalDays:* será un campo obligatorio de tipo entero con un mínimo valor de 1.
- *location:* contendrá las propiedades obligatorias *lat y lng,* que son números y corresponden a las coordenadas de la ubicación del destino.
- *categories:* será un arreglo obligatorio de *strings,* que contiene las preferencias del turista de acuerdo con los tipos de Place Search [37].

2.2.3 Sprint Review

El objetivo del primer *sprint* se lo alcanzó con facilidad, no apareció ningún obstáculo durante el desarrollo gracias a la fácil integración del algoritmo de *clustering* con la API de Google Maps.

Pruebas de aceptación

Tras revisar el trabajo realizado en el primer *sprint,* en la Tabla 7 se puede verificar que se cumplieron todos los criterios de aceptación de las historias de usuario.

Tabla 7. Pruebas de aceptación del Sprint 1

Historia de Usuario	Criterio de aceptación	Cumplido
AH1-01	El servicio deberá extraer sitios turísticos cercanos a la ubicación de destino del turista.	SÍ
	Los sitios turísticos se deben ajustar a los gustos del turista.	SÍ
AH1-02	Dado un conjunto de puntos de interés, cuando se los agrupe geográficamente, las distancias entre los puntos de cada sector deben ser relativamente cortas.	SÍ
	El número de sectores geográficos debe equivaler al número de días de visita del turista.	SÍ

Incremento obtenido

Flujo normal

En la Figura 19 se observa el cuerpo de petición para consumir el servicio REST del primer incremento. La ubicación será dentro de la ciudad de Quito, se estiman 3 días de visita y de preferencias turísticas se seleccionaron museos, iglesias y parques.

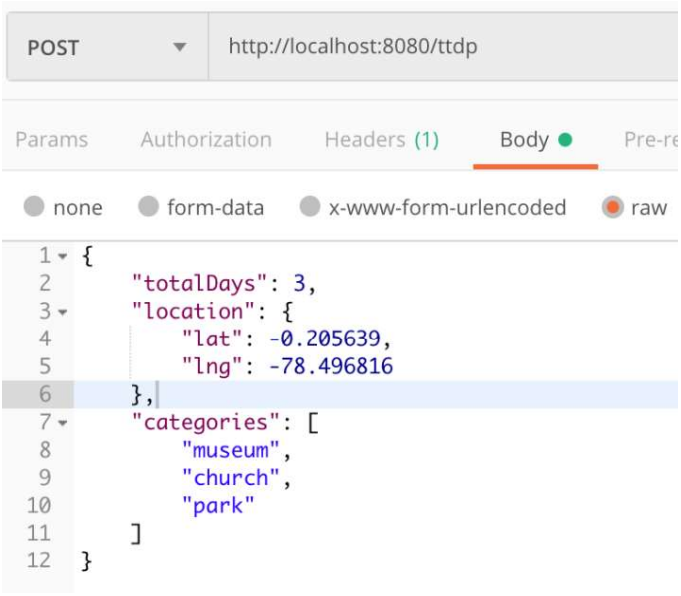

Figura 19. Cuerpo de petición para el primer incremento (Cliente Postman)

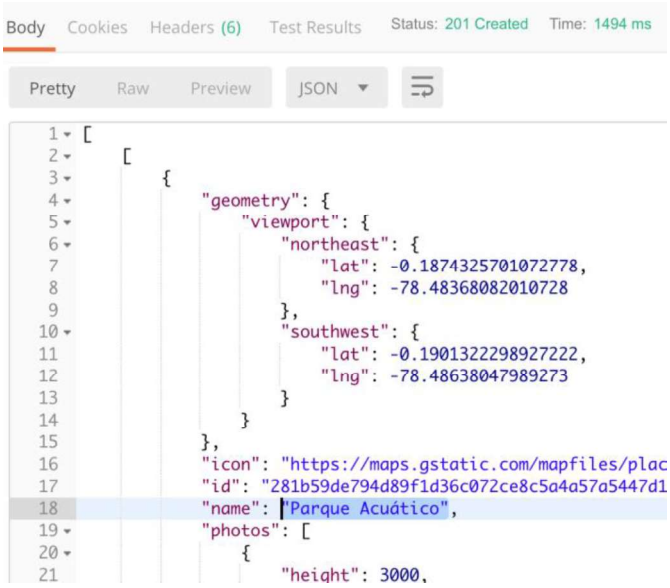

Figura 20. Respuesta del servidor en el primer incremento

La respuesta del servidor (Figura 20) es un objeto json que contiene un arreglo de 3 elementos, cada arreglo corresponde a un grupo de puntos dentro de un sector geográfico (Figura 21). En la Figura 22 se puede apreciar las agrupaciones geográficas dibujadas en un mapa, cada grupo tendrá un color diferente.

Figura 21. Interpretación del resultado obtenido en el incremento 1

Figura 22. Agrupaciones geográficas calculadas con *k-means-same-size*

Flujo con error

En caso de enviar un cuerpo de petición inválido (Figura 23), el servidor responde un error HTTP 400 y en los detalles de la respuesta se especifica el campo faltante (Figura 24).

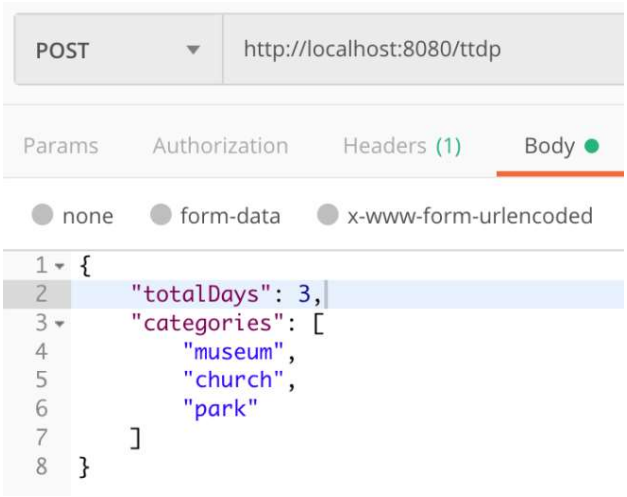

Figura 23. Cuerpo de petición erróneo para el incremento 1 (no se proporciona la ubicación geográfica del destino)

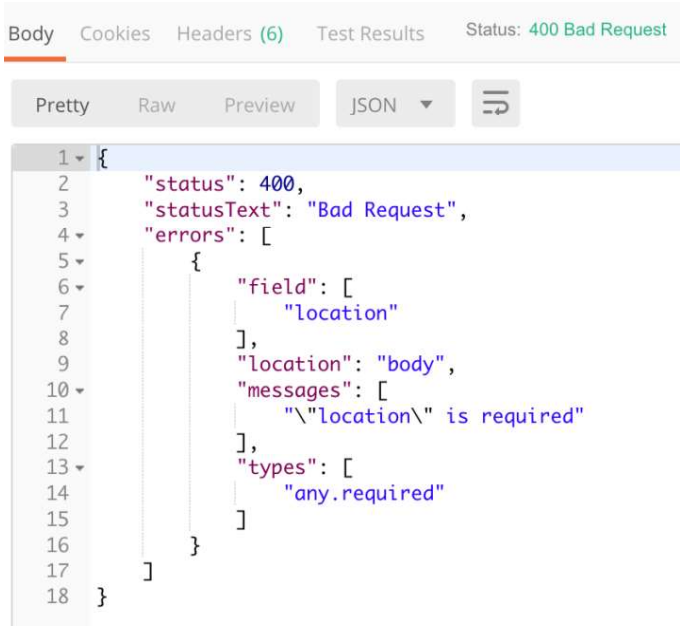

Figura 24. Respuesta de tipo HTTP 400 del servidor

Adaptación del Product Backlog

Tras realizar un análisis del incremento obtenido y sobre el trabajo restante por hacer, se concluyó que todavía no se dispone de información esencial para crear los itinerarios turísticos. Por lo tanto, se agregó otra historia más en el product backlog para obtener los horarios de atención y tiempos de movilización entre los diferentes lugares. En la Tabla 9, se puede ver cómo se adaptó la pila de producto tras la revisión del primer sprint.

Tabla 8. Product Backlog adaptado tras finalizar el Sprint 1

PRODUCT BACKLOG				
HISTORIA ÉPICA	**HISTORIA DE USUARIO**			
	ID	**Título**	**Estimación (días)**	**Estado**
AH1	AH1-01	Extracción de los puntos de interés del turista.	~~5~~	Terminado
	AH1-02	Agrupación geográfica de los POIs.	~~9~~	Terminado
AH2	AH2-01	Generación de itinerarios turísticos optimizados.	13	Por implementar
	AH2-02	Horarios de atención y tiempos de viaje de los POIs	1	Por implementar
AH3	AH3-01	Desarrollo del formulario del usuario final.	8	Por implementar
	AH3-02	Desarrollo de la interfaz de despliegue de resultados.	6	Por implementar

2.2.4 Sprint Retrospective

En la Figura 25, se observa el gráfico de *Burndown* donde se representa el progreso del trabajo durante todo el *sprint.* Como se puede apreciar, el conocimiento y la experiencia del equipo de desarrollo permitió que el producto se genere en menor tiempo del estimado.

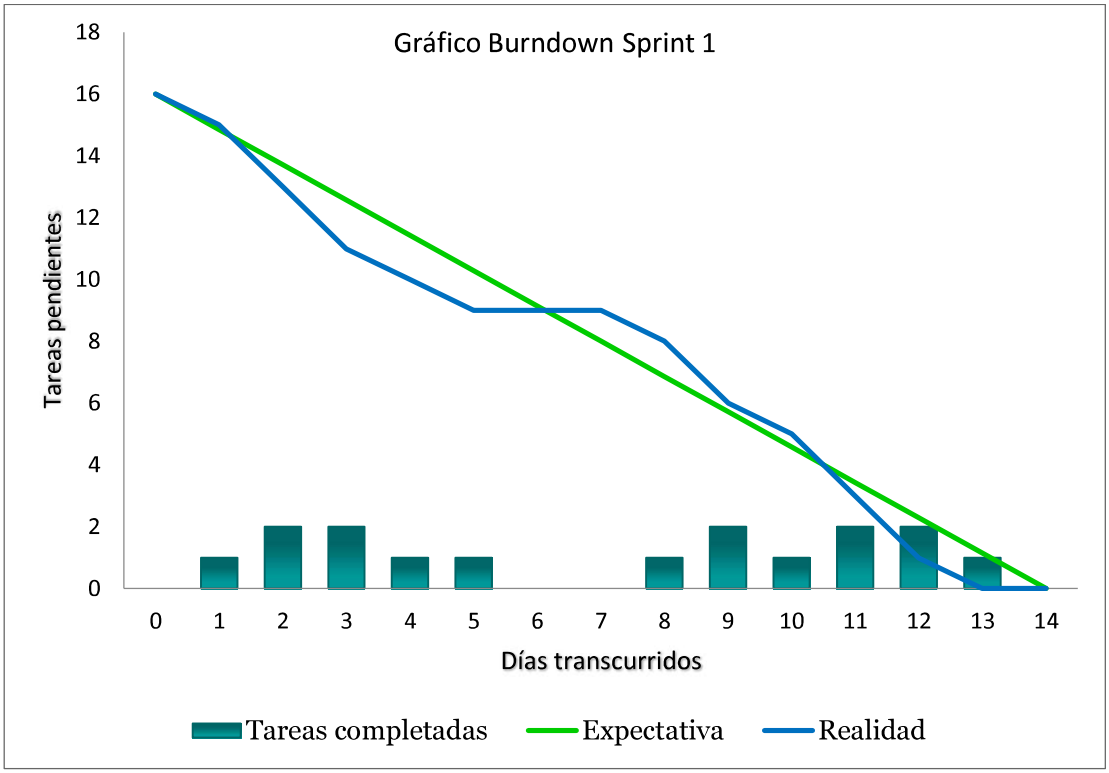

Figura 25. Gráfico de Burndown del Sprint 1

2.3 Sprint 2

2.3.1 Sprint Planning

Objetivo del Sprint

Completar las fases de inicialización, selección, *crossover* y mutación del AG personalizado con el fin de optimizar el tiempo del turista durante su viaje.

Historias de Usuario

Tabla 9. Historias de usuario escogidas para el Sprint 2 (Sus detalles se encuentran en el Anexo III)

ID	Título	Estimación (días)	Estado
AH2-02	Horarios de atención y tiempos de viaje de los POIs.	1	Por implementar
AH2-01	Generación de itinerarios turísticos optimizados.	13	Por implementar

Sprint Backlog

Tabla 10. Sprint Backlog del Sprint 2

SPRINT BACKLOG	
HISTORIA DE USUARIO	**TAREAS**
AH2-02	Modificar el cuerpo de petición para aceptar la fecha inicial de los tours.
	Obtener los horarios de atención de cada POI.
	Eliminar los POIs que no estén disponibles durante las fechas de viaje.
	Agregar la ubicación de hospedaje del turista dentro de cada uno de los clústeres obtenidos.
	Consumir la Directions API de Google para obtener el tiempo estimado de recorrido entre los diferentes puntos de cada clúster
	Armar una matriz de tiempos con los resultados obtenidos.
AH2-01	Creación del proyecto base para el AG.
	Crear una clase *javascript* llamada *GeneticAlgorithm* que reciba en su constructor los parámetros requeridos.
	Codificar las soluciones mediante permutaciones de secuencia de números.
	Generar una población de cromosomas aleatorios de acuerdo con el tamaño recibido en los parámetros en el constructor.
	Realizar pruebas unitarias que validen la instanciación de la clase y la codificación de las soluciones.
	Desarrollar la función fitness que evaluará la calidad de las soluciones
	Implementar el algoritmo de selección por ruleta.
	Implementar el algoritmo de *Partially-mapped crossover.*
	Desarrollar un algoritmo simple de mutación que cambie las posiciones dentro de un cromosoma.
	Generar los itinerarios turísticos.

2.3.2 Ejecución del Sprint

Durante el segundo *sprint* se pueden rescatar los siguientes puntos:

Google Maps API

Se tuvieron que consumir 2 *enpoints* más para obtener la información que se requería en los parámetros del AG. La extracción de los horarios de atención y otros detalles extra se puede observar en el diagrama de secuencia de la Figura 26, se tuvo que iterar cada clúster y consumir el *endpoint Details* de la *Places API.* Además, se descartaron los sitios que no atendían durante la fecha del tour.

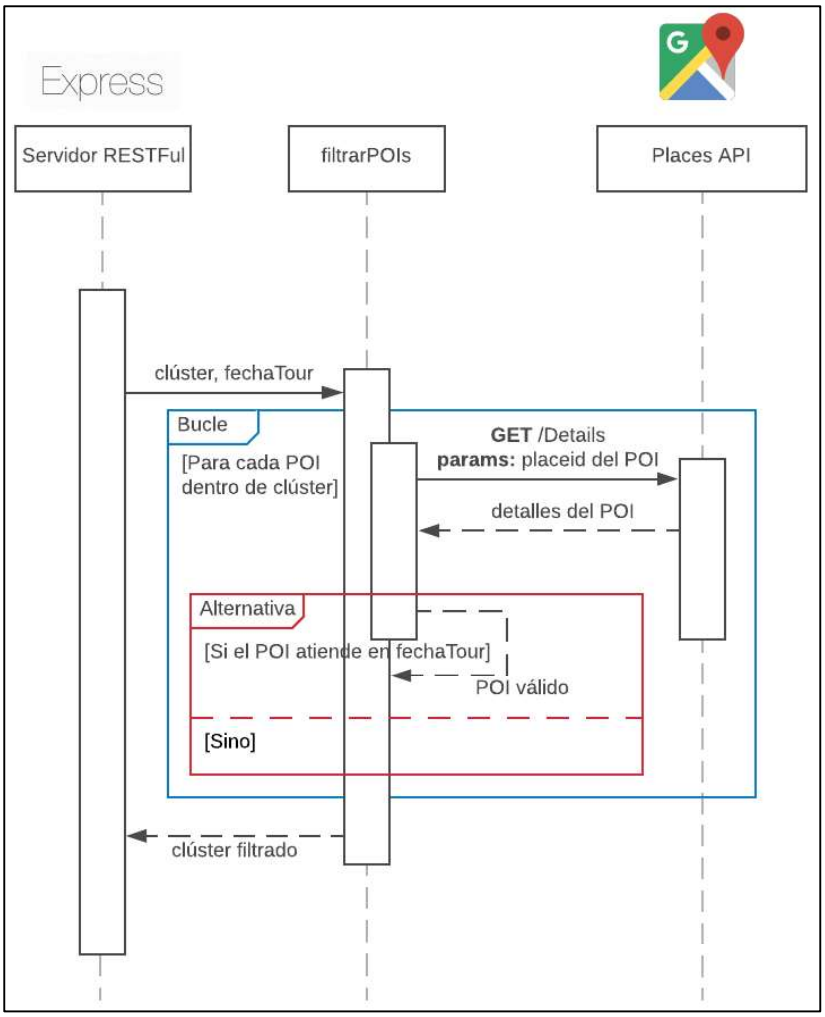

Figura 26. Diagrama de secuencia para obtener los detalles de cada POI dentro de un clúster

Para obtener los tiempos de recorrido entre los diferentes puntos del clúster (incluyendo el lugar de hospedaje), se consumió la *Directions API* de *Google* en una iteración que permita llenar la matriz de tiempos (ver Figura 27).

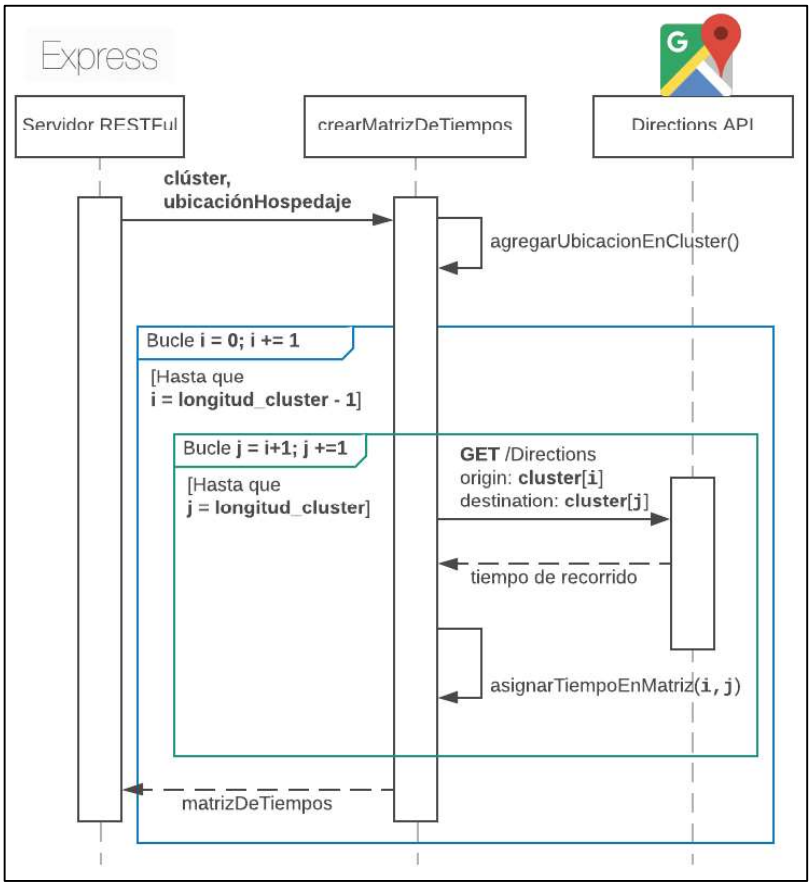

Figura 27. Diagrama de secuencia para obtener la matriz de tiempos

Parámetros del AG

El AG se lo plasmó en una clase de *Javascript* que acepta los siguientes parámetros en su constructor:

Parámetros exógenos:

- El tamaño de la población (número).
- El número total de generaciones (número).
- La probabilidad de crossover (número).

Parámetros endógenos (utilizados en la función *fitness*):

- La fecha del día del tour (objeto).
- Un clúster (arreglo de objetos) que contiene el lugar de hospedaje y los puntos de interés con su información detallada (horarios de atención y tiempo estimado de visita), como se muestra en la Figura 28.

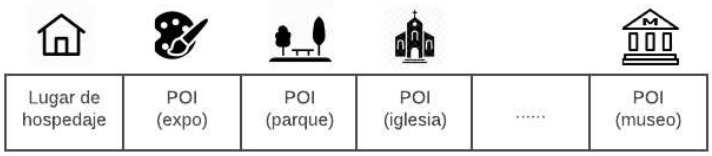

Figura 28. Ejemplo de un clúster

- Una matriz de tiempos de desplazamiento entre los diferentes puntos del clúster (matriz de objetos) como la Figura 29.

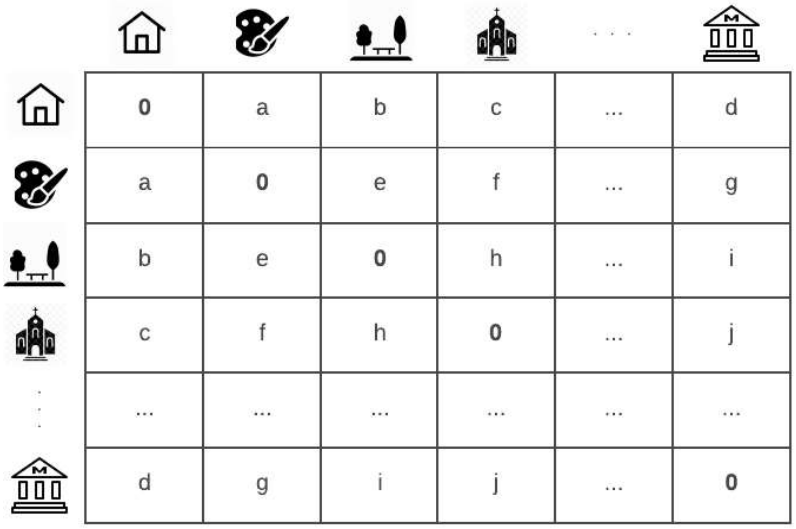

Figura 29. Matriz de tiempos entre los diferentes puntos (cada letra representa el tiempo de movilización entre dos puntos)

Codificación de las soluciones

Se codificó a cada cromosoma mediante permutaciones de números, en la Figura 30 se observa un ejemplo.

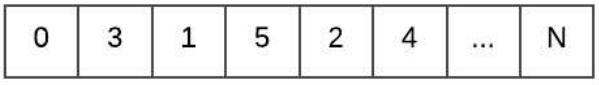

Figura 30. Ejemplo de una solución codificada (cromosoma)

Cada gen del cromosoma (elemento de la solución codificada) equivale a una posición dentro del clúster; de modo que, si se itera la solución en orden, se obtendrá un tour (ver Figura 31).

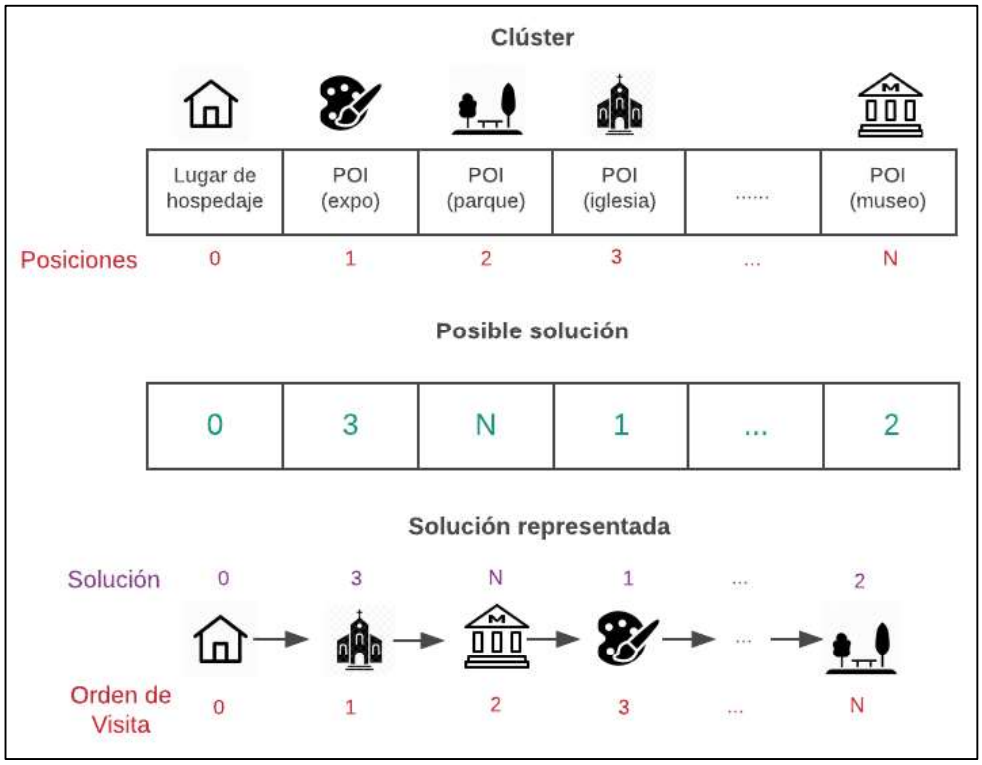

Figura 31. Representación de una posible solución

Función fitness

Se evaluó la factibilidad de las soluciones basándose en métricas tales como el tiempo de recorrido entre los distintos lugares, sus horarios de atención, la hora de almuerzo y el tiempo estimado para visitar cada lugar. Por lo tanto, se implementó el algoritmo de las Figuras 32, 33 y 34 que calcula el tiempo total que le tomaría al turista visitar los puntos de acuerdo con el orden sugerido para luego compararlo con el tiempo disponible del tour.

Se asumió que cada tour tendrá 8 horas de duración, que empieza a las 9 am. y que la hora del almuerzo será entre la 1 pm. y 2 pm. Al trabajar con horarios, se utilizó un objeto de tipo *Date* denominado **horaActual** que registra el tiempo de cualquier evento (visitar un POI o moverse al siguiente) desde que inicia el tour. Además, se añadió penalizaciones de tiempo en caso de llegar fuera de los horarios de atención de algún POI; y también se consideró si el itinerario perjudica la hora destinada para el almuerzo.

```
SubProceso calcularFitness(cromosoma, cluster, matrizDeTiempo)
  horaActual <- 09H00
  tiempoDisponible <- 8 horas
  longitudRuta <- cromosoma.longitud
  // Tiempo de viaje desde el hospedaje hacia el primer POI:
  tiempoViaje <- matrizDeTiempo[cromosoma[0]][cromosoma[1]].tiempo_viaje
  tiempoInvertido <- tiempoViaje

  i <- 1
  Mientras i < longitudRuta - 1
  Hacer
    poiActual <- cluster[cromosoma[i]]
    penalizaciones <- calcularPenalizaciones(horaActual, poiActual.horarios, tiempoViaje)
    // Tiempo de visita del POI actual:
    tiempoVisita <- poiActual.tiempo_visita
    duracionVisita <- ejecutarEvento(horaActual, tiempoVisita, horarioAlmuerzo)
    // Tiempo de recorrido hacia el siguiente POI:
    tiempoViaje <- matrizDeTiempo[cromosoma[i]][cromosoma[i+1]].tiempo_viaje
    duracionViaje <- ejecutarEvento(horaActual, tiempoViaje, horarioAlmuerzo)

    tiempoInvertido <- tiempoInvertido + penalizaciones + duracionVisita + duracionViaje
    i <- i + 1
  Fin Mientras

  ultimoPOI <- cluster[cromosoma[longitudRuta-1]]
  penalizaciones <- calcularPenalizaciones(horaActual, ultimoPOI.horarios, tiempoDeViaje)
  tiempoInvertido <- tiempoInvertido + penalizaciones + ultimoPOI.tiempo_visita

  retornar |tiempoDisponible - tiempoInvertido|

Fin SubProceso
```

Figura 32. Seudocódigo general de la función fitness

Las penalizaciones se calculan basándose en la puntualidad, en caso de llegar a un POI dentro de sus horarios de atención, no hay penalización; si se llega antes, la penalización será el tiempo que toque esperar hasta que el sitio abra; pero en caso de llegar después, la penalización será el tiempo de viaje que tomó llegar hacia dicho lugar.

```
SubProceso calcularPenalizaciones(horaActual, horarioAtencion, tiempoDeViaje)
      horaApertura <- horarioAtencion.apertura
      horaCierre <- horarioAtencion.cierre
      Si horaActual se encuentra entre horaApertura y horaCierre
      Entonces
            retornar 0
      Sino
      Entonces
            Si horaActual está antes de horaApertura
            Entonces
                  retornar horaApertura - horaActual
            Sino horaActual está después de horaCierre
            Entonces
                  retornar tiempoDeViaje
            Fin si
      Fin Si
Fin SubProceso
```

Figura 33. Seudocódigo de la función utilizada para calcular las penalizaciones de tiempo.

También se consideró una métrica llamada “tiempo invadido”, la cual corresponde al tiempo que desfasa el horario planificado para el almuerzo (1 pm - 2 pm). Es decir, si un evento

se cruza con el horario de almuerzo y se termina almorzando a la 1:20 pm, el tiempo invadido equivaldrá a 20 minutos. Mientras menor sea ese tiempo invadido, mejor será la calidad de la solución.

```
SubProceso ejecutarEvento(horaActual, tiempoEvento)
  inicioAlmuerzo <- 1:00pm
  tiempoAlmuerzo <- 60 minutos
  tiempoTotal <- tiempoEvento
  siguienteHora <- horaActual + tiempoEvento

  Si inicioAlmuerzo está entre horaActual y siguienteHora
  Entonces
    horaActual <- horaActual + tiempoAlmuerzo + tiempoEvento

    Si |inicioAlmuerzo - horaActual| es menor |siguienteHora - inicioAlmuerzo|
    Entonces
      tiempoInvadido <- inicioAlmuerzo - horaActual
    Sino
    Entonces
      tiempoInvadido <- siguienteHora - inicioAlmuerzo
    Fin si
    tiempoTotal <- tiempoTotal + tiempoAlmuerzo + tiempoInvadido
  Sino
  Entonces
    horaActual <- horaActual + tiempoEvento
  Fin si
  retornar tiempoTotal
Fin SubProceso
```

Figura 34. Seudocódigo de la función utilizada para medir el tiempo invertido en la ejecución de algún evento, considerando el horario de comida.

Fase de selección del AG

En la solución propuesta, el valor fitness de cada cromosoma corresponde a la diferencia entre el tiempo invertido y el tiempo disponible del tour. Mientras más pequeña sea esa diferencia, la solución es mejor. En la fase de selección se implementó el algoritmo de selección por ruleta, que brinda mejores probabilidades de selección a los cromosomas que tengan un valor *fitness* más alto. Por lo tanto, antes de la selección se realizó el ajuste que se muestra en la ecuación 2.

$$fitnessAjustado = \frac{1}{FitnessActual}$$

Ecuación 2. Fórmula para el ajuste del fitness

Al invertir su valor se asegura que los cromosomas con fitness más bajos tendrán una mejor probabilidad de ser seleccionados en la ruleta.

Fase de mutación

Se implementó un algoritmo de mutación sencillo que se puede visualizar en la Figura 35. Consiste en seleccionar un conjunto aleatorio de genes, agruparlos en parejas e intercambiar sus lugares.

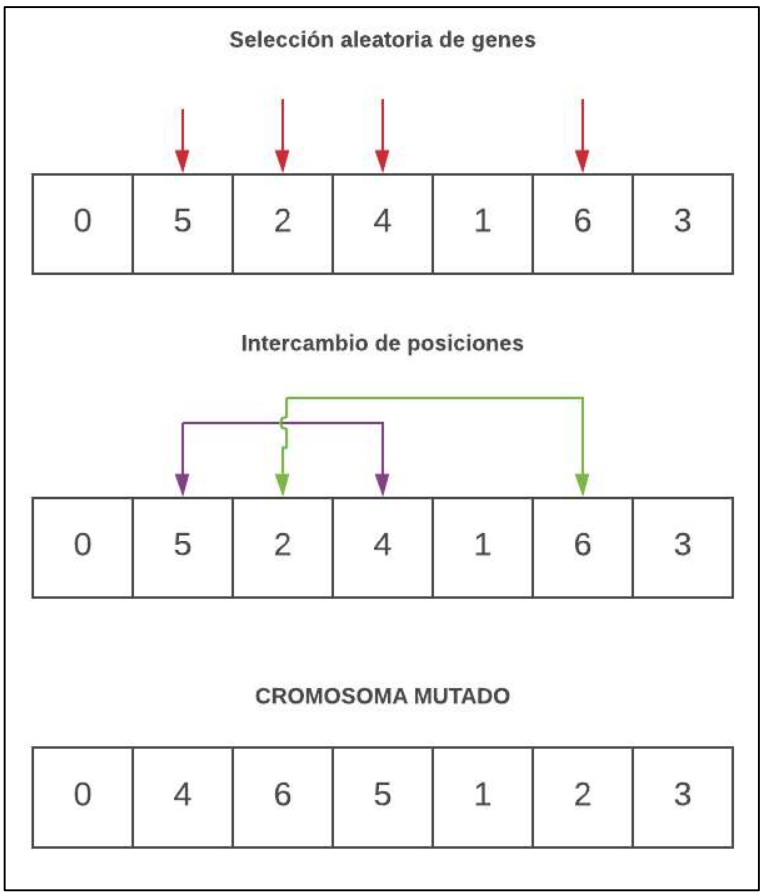

Figura 35. Algoritmo de mutación implementado

Pruebas unitarias

Se utilizó las librerías de Mocha y Chai para crear un ambiente de pruebas fácil y flexible que permita comprobar la funcionalidad del código. En las Figuras 36 y 37 se puede observar un par de ejemplo de pruebas unitarias.

```
it('should init population', (done) => {
    ga = new GeneticAlgorithm({
        pois: mockPois(),
        timeMatrix: mockMatrix(),
        travelDate:  new Date( year: 2019,  month: 1 ,  date: 6),
        totalGenerations: 100,
        crossOverProbability: 1/3,
        populationSize: 100
    });
    expect(ga.population).to.have.length(100);
    done();
});
```

Figura 36. Prueba unitaria para verificar la inicialización de una población.

```
it('should select parents to crossover', (done) => {
    ga = new GeneticAlgorithm({
        pois: mockPois(),
        timeMatrix: mockMatrix(),
        travelDate:  new Date( year: 2019,  month: 1 ,  date: 6),
        totalGenerations: 100,
        crossOverProbability: 1/3,
        populationSize: 100
    });

    const expected_size = Math.floor( x: ga.population.length *
                                    parameters.crossOverProbability);
    const selected = ga.selection();

    expect(selected.length).to.be.eq(expected_size);
    done();
});
```

Figura 37. Prueba unitaria para verificar el número de cromosomas seleccionados

2.3.3 Sprint Review

Pruebas de aceptación

En la Tabla 11 se listan los criterios de aceptación cumplidos para el segundo *sprint*. En este caso, faltó por cubrir sólo un criterio, el cual fue completado con facilidad en el último *sprint.*

Tabla 11. Pruebas de aceptación del Sprint 2

Historia de Usuario	Criterio de aceptación	Cumplido
AH2-02	Cada punto de interés debe contener información sobre sus horarios de atención.	SÍ
	Se debe descartar los sitios que no atiendan durante los días de visita del turista.	SÍ
	Se debe conocer el tiempo de movilización (a pie) entre los puntos de un sector geográfico.	SÍ
AH2-01	Dado un grupo geográfico de puntos turísticos, cuando se genere un itinerario óptimo, se deberá saber el orden de cómo visitarlos.	NO
	El itinerario debe tener en cuenta el horario de almuerzo entre la 1pm y 2pm de la tarde.	SÍ
	El itinerario debe considerar los horarios de atención y el tiempo de movilización entre los diferentes puntos.	SÍ
	El itinerario debe tener un tiempo de recorrido de aproximadamente 8 horas.	SÍ

Incremento obtenido

Clústeres filtrados

En la Figura 38 se incluye el nuevo parámetro que corresponde a la fecha de inicio de los tours. Entonces, a cada clúster se le asignó una respectiva fecha que permitirá filtrar los puntos de interés que no atiendan en el día destinado para realizar el recorrido; también se agregó la ubicación de hospedaje con el fin de incluir ese punto al momento de calcular la matriz de tiempos (Figura 39).

```
{
    "totalDays": 3,
    "startDate": "2019-01-06",
    "location": {
        "lat": -0.205639,
        "lng": -78.496816
    },
    "categories": [
        "museum",
        "church",
        "park"
    ]
}
```

Figura 38. Cuerpo de solicitud que incluye la fecha de inicio de viaje

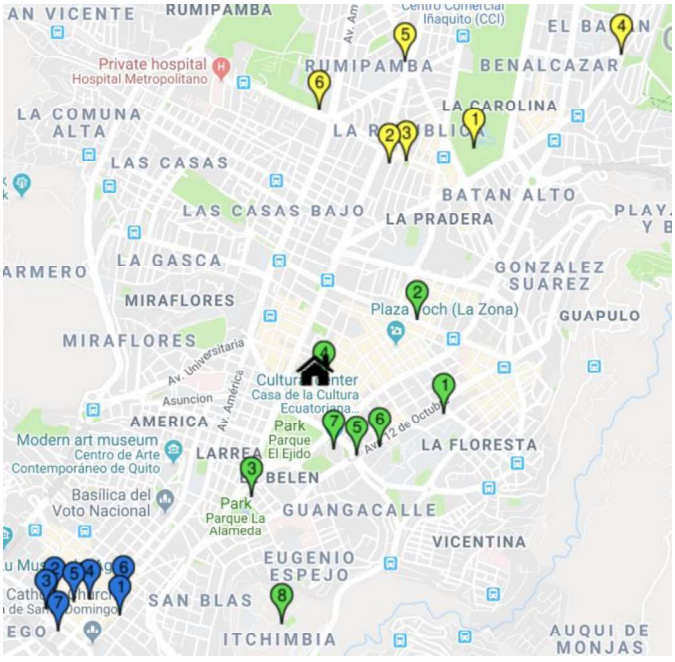

Figura 39. Clústeres filtrados

Algoritmo genético

La Figura 40 es un pequeño bloque de código que muestra la instancia de la clase del AG. Los resultados se observan en la Figura 41, donde se verifica que el tamaño de la población inicializada coincide con el parámetro proporcionado en el constructor, las soluciones están correctamente codificadas y en el proceso de selección se escogió al 30% de cromosomas, equivalente a la probabilidad de *crossover* fijada en los parámetros.

```
const ga = new GeneticAlgorithm({
    pois: [...],
    timeMatrix: [...],
    travelDate: new Date( year: 2019,  month: 0,  date: 6),
    totalGenerations: 200,
    crossOverProbability: 0.3,
    populationSize: 100
});

console.log("--RESULTADOS--");
console.log("Tamaño de la población:", ga.population.length);
console.log("Ejemplo de cromosoma:", ga.population[0].chromosome);

const parents = ga.selection();
console.log("Cromosomas seleccionados:", parents.length);
```

Figura 40. Pequeño bloque de código para ejecutar el incremento del Sprint 2

```
--RESULTADOS--
Tamaño de la población: 100
Ejemplo de cromosoma: [ 0, 1, 4, 3, 6, 5, 2 ]
Cromosomas seleccionados: 30
```

Figura 41. Resultados del algoritmo genético en el Sprint 2.

Adaptación del Product Backlog

Después de realizar un análisis entre los miembros del equipo, se concluyó que es necesario parametrizar los horarios de los tours y la hora de almuerzo de acuerdo con las preferencias propias del turista; además, no se tomó en cuenta incluir recomendaciones de restaurantes durante la hora de comida; y también faltó por concluir un par de tareas para la historia AH2-01. Por ende, se procedió a adaptar al *product backlog* como se enseña en la Tabla 12.

Tabla 12. Product Backlog adaptado tras finalizar el Sprint 2

PRODUCT BACKLOG				
HISTORIA ÉPICA	**HISTORIA DE USUARIO**			
	ID	**Título**	**Estimación (días)**	**Estado**
AH1	AH1-01	Extracción de los puntos de interés del turista.	~~5~~	Terminado
	AH1-02	Agrupación geográfica de los POIs.	~~9~~	Terminado
AH2	AH2-01	Generación de itinerarios turísticos optimizados.	2	En desarrollo
	AH2-02	Horarios de atención y tiempos de viaje de los POIs	~~1~~	Terminado
	AH2-03	Parametrización de los horarios de tour y de almuerzo.	2	Por implementar
	AH2-04	Recomendaciones de restaurantes para el horario de comida.	1	Por implementar
AH3	AH3-01	Desarrollo del formulario del usuario final.	5	Por implementar
	AH3-02	Desarrollo de la interfaz de despliegue de resultados.	4	Por implementar

2.3.4 Sprint Retrospective

La Figura 42 corresponde al gráfico de *burndown* del segundo sprint. En este caso, el equipo presentó dificultades durante el desarrollo por la difícil implementación de ciertos algoritmos para las fases del AG. Por tal razón, en los días 7, 8 y 9 se presentó un estancamiento en la realización de las tareas. Aunque faltó por concluir la generación de itinerarios, no representó ningún obstáculo para el siguiente *sprint.*

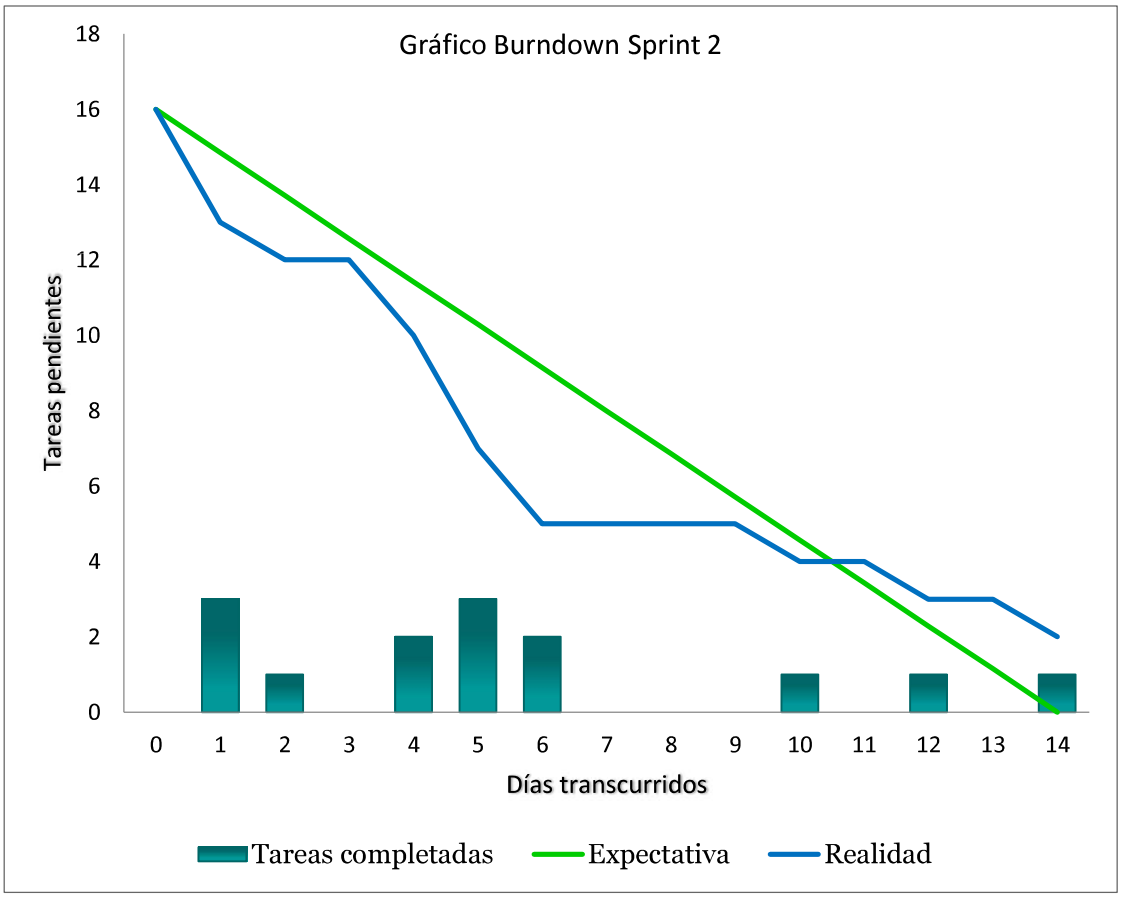

Figura 42. Gráfico de Burndown del Sprint 2

2.4 Sprint 3

2.4.1 Sprint Planning

Objetivo del Sprint

Concluir el desarrollo del AG e implementar una aplicación de *front-end* sencilla para representar el resultado obtenido de acuerdo con el diseño del Anexo V.

Historias de Usuario

Las historias de usuario restantes que se tomaron para el último *sprint* se encuentran listadas en la Tabla 13. En este caso, se transfirió la historia de usuario AH2-01 ya que no se consiguió completar con todos sus criterios de aceptación en el anterior *sprint*.

Tabla 13. Historias de usuario escogidas para el Sprint 3 (Sus detalles completos se encuentran en el Anexo IV)

ID	Título	Estimación (días)	Estado
AH2-01	Generación de itinerarios turísticos optimizados.	2	En desarrollo
AH2-03	Parametrización de los horarios de tour y de almuerzo.	2	Por implementar
AH2-04	Recomendaciones de restaurantes para el horario de comida.	1	Por implementar
AH3-01	Desarrollo del formulario del usuario final.	5	Por implementar
AH3-02	Desarrollo de la interfaz de despliegue de resultados.	4	Por implementar

Sprint Backlog

Tabla 14. Sprint backlog del Sprint 3

SPRINT BACKLOG	
HISTORIA DE USUARIO	**TAREAS**
AH2-01	Terminar la implementación del algoritmo *Partially-mapped crossover*.
	Generar los itinerarios turísticos.
AH2-03	Agregar los nuevos parámetros en el constructor de la clase.
	Refactorizar la función *fitness* para realizar los cálculos con los nuevos parámetros.
AH3-01	Encontrar la posición geográfica del turista cuando se acerque al horario de comida.
	Consumir la API de Google Maps para buscar recomendaciones de restaurantes más cercanos respecto a esa posición.
	Incluir las recomendaciones de comida dentro de cada itinerario.
AH3-01	Inicializar el proyecto de React utilizando el paquete *create-react-app* de *npm.*
AH3-01	Crear el componente del formulario.
	Agregar todos los inputs necesarios de acuerdo con el diseño.
	Utilizar un componente que despliegue el mapa de Google Maps que permita seleccionar un punto en el mapa.
	Controlar los campos para que sean obligatorios antes de enviar el formulario.
AH3-02	Crear el componente de resultados.
	Agregar todos los elementos que conforman el componente de acuerdo con el diseño.
	Consumir el *endpoint /ttdp* del servicio web con los valores proporcionados en el formulario.
	Utilizar enrutamiento para cargar el nuevo componente.
	Dibujar los resultados en el mapa de Google Maps y mostrar los detalles del itinerario mediante una Tabla para cada día de tour.
	Utilizar íconos para diferenciar POIs, restaurantes y lugar de estadía.
	Agregar ventanas para mostrar información detallada de los puntos en el mapa.

2.4.2 Ejecución del Sprint

Cuerpo de petición final

Tras todas las modificaciones que se realizaron durante el desarrollo, el esquema del cuerpo de petición quedó como se muestra en la Figura 43.

```
body: {
    totalDays: joi.number().integer().positive().min(1).required(),
    location: joi.object().keys({
        lat: joi.number().required(),
        lng: joi.number().required(),
    }).required(),
    categories: joi.array().items(joi.string()),
    startDate: joi.date().required(),
    travelSchedule: joi.object().keys({
        start: joi.string().regex(/^[0-9]{4}$/).required(),
        end: joi.string().regex(/[0-9]{4}/).required(),
    }).default({
        start: "0900",
        end: "1800"
    }),
    lunchTime: joi.object().keys({
        start: joi.string().regex(/[0-9]{4}/).required(),
        end: joi.string().regex(/[0-9]{4}/).required(),
    }).default({
        start: "1300",
        end: "1400"
    })
}
```

Figura 43. Cuerpo de petición final

Se agregaron los siguientes atributos que serán utilizados en el AG:

- *startDate:* es un campo obligatorio que corresponde a la fecha estimada para iniciar los tours en el destino turístico.
- *travelSchedule:* es el horario deseado para realizar los tours, se lo representa con un objeto que tiene las propiedades *start* y *end,* las cuales son *strings* que cumplen con una expresión regular que representa una hora en específico. En caso de no especificar este campo, los valores por defecto serán entre las 09H00 y las 16H00.
- *lunchTime:* de igual manera, es un objeto que representa el horario deseado para el almuerzo y sus valores por defecto son entre las 13H00 y 14H00

Función fitness

Tras parametrizar los horarios de tours y de almuerzo, el algoritmo de la función fitness se modificó de acuerdo con el seudocódigo expuesto en las Figuras 44 y 45.

```
SubProceso calcularFitness(cromosoma, cluster, matrizDeTiempo, horarioTour, horarioAlmuerzo)
  horaActual <- horarioTour.inicio
  tiempoDisponible <- horarioTour.fin - horarioTour.inicio
  longitudRuta <- cromosoma.longitud
  tiempoViaje <- matrizDeTiempo[cromosoma[0]][cromosoma[1]].tiempo_viaje
  tiempoInvertido <- tiempoViaje

  i <- 1
  Mientras i < longitudRuta - 1
  Hacer
    poiActual <- cluster[cromosoma[i]]
    penalizaciones <- calcularPenalizaciones(horaActual, poiActual.horarios, tiempoViaje)
    tiempoVisita <- poiActual.tiempo_visita
    duracionVisita <- ejecutarEvento(horaActual, tiempoVisita, horarioAlmuerzo)
    tiempoViaje <- matrizDeTiempo[cromosoma[i]][cromosoma[i+1]].tiempo_viaje
    duracionViaje <- ejecutarEvento(horaActual, tiempoViaje, horarioAlmuerzo)

    tiempoInvertido <- tiempoInvertido + penalizaciones + duracionVisita + duracionViaje
    i <- i + 1
  Fin Mientras

  ultimoPOI <- cluster[cromosoma[longitudRuta-1]]
  penalizaciones <- calcularPenalizaciones(horaActual, ultimoPOI.horarios, tiempoDeViaje)
  tiempoInvertido <- tiempoInvertido + penalizaciones + ultimoPOI.tiempo_visita

  retornar |tiempoDisponible - tiempoInvertido|

Fin SubProceso
```

Figura 44. Seudocódigo final de la función fitness que parametriza los horarios de tour y de almuerzo.

```
SubProceso ejecutarEvento(horaActual, tiempoEvento, horarioAlmuerzo)
  inicioAlmuerzo <- horarioAlmuerzo.inicio
  tiempoAlmuerzo <- horarioAlmuerzo.fin - horarioAlmuerzo.inicio
  tiempoTotal <- tiempoEvento
  siguienteHora <- horaActual + tiempoEvento

  Si inicioAlmuerzo está entre horaActual y siguienteHora
  Entonces
    horaActual <- horaActual + tiempoAlmuerzo + tiempoEvento

    Si |inicioAlmuerzo - horaActual| es menor |siguienteHora - inicioAlmuerzo|
    Entonces
      tiempoInvadido <- inicioAlmuerzo - horaActual
    Sino
    Entonces
      tiempoInvadido <- siguienteHora - inicioAlmuerzo
    Fin si
    tiempoTotal <- tiempoTotal + tiempoAlmuerzo + tiempoInvadido
  Sino
  Entonces
    horaActual <- horaActual + tiempoEvento
  Fin si
  retornar tiempoTotal
Fin SubProceso
```

Figura 45. Seudocódigo final de la función utilizada para medir el tiempo invertido en la ejecución de un evento considerando el horario de almuerzo parametrizado.

En las Figuras se puede apreciar con color rojo las modificaciones que se realizaron para tomar en cuenta los horarios de tour y almuerzo ajustados de acuerdo con las preferencias del turista. De esta manera, no se realizan suposiciones y las predicciones serán más personalizadas.

Diagrama de secuencia final

El flujo final se puede apreciar en el diagrama de la Figura 46, empieza con una solicitud del turista desde un navegador web donde proporciona sus preferencias de viaje en un formulario; con éstas, se procede a consumir la API de Google para obtener un conjunto de POIs, del cual se calcularán los clústeres geográficos utilizando el algoritmo *kmeans.* Para cada clúster, se procede a filtrar los sitios que no atiendan durante el día del tour; a calcular la matriz de tiempos de viaje entre el lugar de hospedaje y los diferentes POIs; a generar un itinerario optimizado con el AG; y a obtener recomendaciones de restaurantes. Por último, se dibujan los resultados en una interfaz amigable donde el turista podrá visualizar las recomendaciones que se generaron.

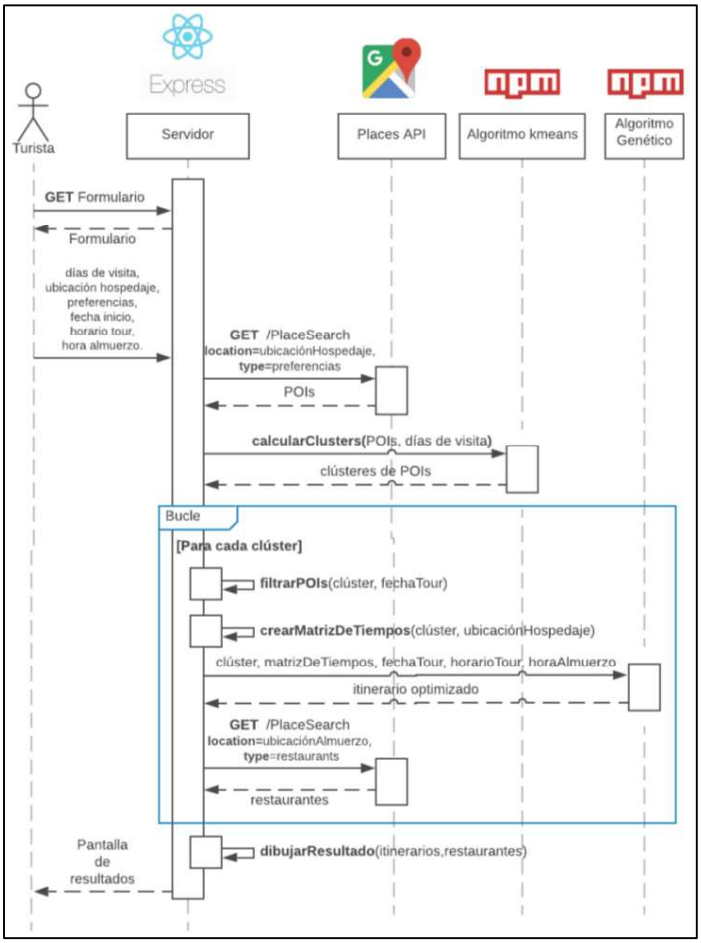

Figura 46. Diagrama de secuencia del producto final.

2.4.3 Sprint Review

Pruebas de aceptación

Historia de Usuario	Criterio de aceptación	Cumplido
AH2-01	Dado un grupo geográfico de puntos turísticos, cuando se genere un itinerario óptimo, se deberá saber el orden de cómo visitarlos.	SÍ
	El itinerario debe tener en cuenta el horario de almuerzo fijado por el turista.	SÍ
	El itinerario debe considerar los horarios de atención y el tiempo de movilización entre los diferentes puntos.	SÍ
	El itinerario debe tener un tiempo de recorrido establecido por el turista.	SÍ
AH2-03	Los itinerarios generados deberán acoplarse con los horarios establecidos por el turista.	SÍ
AH2-04	Los restaurantes recomendados deben estar cerca de la posición del turista cuando caiga la hora de almuerzo.	SÍ
AH3-01	El formulario debe recopilar información como días de recorrido, fecha de inicio de viaje, horarios deseados para tours y almuerzo, preferencias turísticas y lugar de hospedaje.	SÍ
	El lugar de hospedaje debe ser seleccionado en un mapa interactivo.	SÍ
	Cuando el turista envíe la información completa de sus preferencias, entonces debería aparecer una pantalla de carga.	SÍ
	Cuando el turista lo intente enviar su información incompleta, se deberán mostrar los campos que faltan por llenar.	SÍ
AH3-02	La pantalla deberá desplegar los itinerarios en un mapa y trazar las rutas entre los diferentes POIs.	SÍ
	Cuando se de clic sobre un POI, se deberá abrir un cuadro con información detallada	SÍ
	El turista deberá poder seleccionar el tour de un día en específico.	SÍ
	Para cada día, se deberá mostrar un cuadro con la información detallada de todo el itinerario.	SÍ

Adaptación del Product Backlog

Tabla 15. Product backlog tras la finalización del proyecto

PRODUCT BACKLOG				
HISTORIA ÉPICA	**HISTORIA DE USUARIO**			
	ID	**Título**	**Estimación (días)**	**Estado**
AH1	AH1-01	Extracción de los puntos de interés del turista.	~~5~~	Terminado
	AH1-02	Agrupación geográfica de los POIs.	~~9~~	Terminado
AH2	AH2-01	Generación de itinerarios turísticos optimizados.	~~2~~	Terminado
	AH2-02	Horarios de atención y tiempos de viaje de los POIs	~~1~~	Terminado
	AH2-03	Parametrización de los horarios de tour y de almuerzo.	~~2~~	Terminado
	AH2-04	Recomendaciones de restaurantes para el horario de comida.	~~1~~	Terminado
AH3	AH3-01	Desarrollo del formulario del usuario final.	~~5~~	Terminado
	AH3-02	Desarrollo de la interfaz de despliegue de resultados.	~~4~~	Terminado

2.4.4 Sprint Retrospective

En la última reunión de retrospectiva, no se tuvo ninguna observación ya que se logró cumplir con todo el trabajo planificado dentro del tiempo establecido; en el gráfico de *burndown* de la Figura 47 se puede observar cómo se distribuyó el tiempo de una mejor manera para concluir con todas las tareas del proyecto.

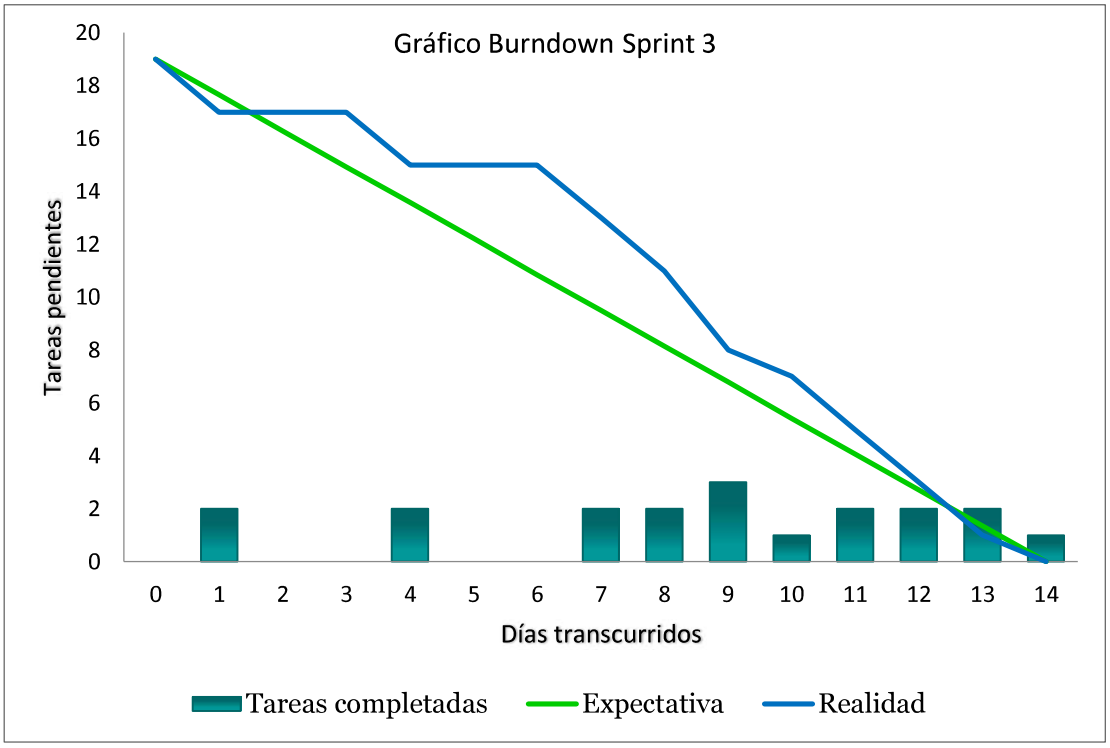

Figura 47. Gráfico *burndown* del *Sprint* 3

3 RESULTADOS Y DISCUSIÓN

3.1 Producto Final

El producto final del actual proyecto consta de tres pantallas: una para la de toma de datos, otra para el despliegue de resultados y una adicional que se muestra durante la transición de ambas, mientras se procesan los datos (ver Figura 48).

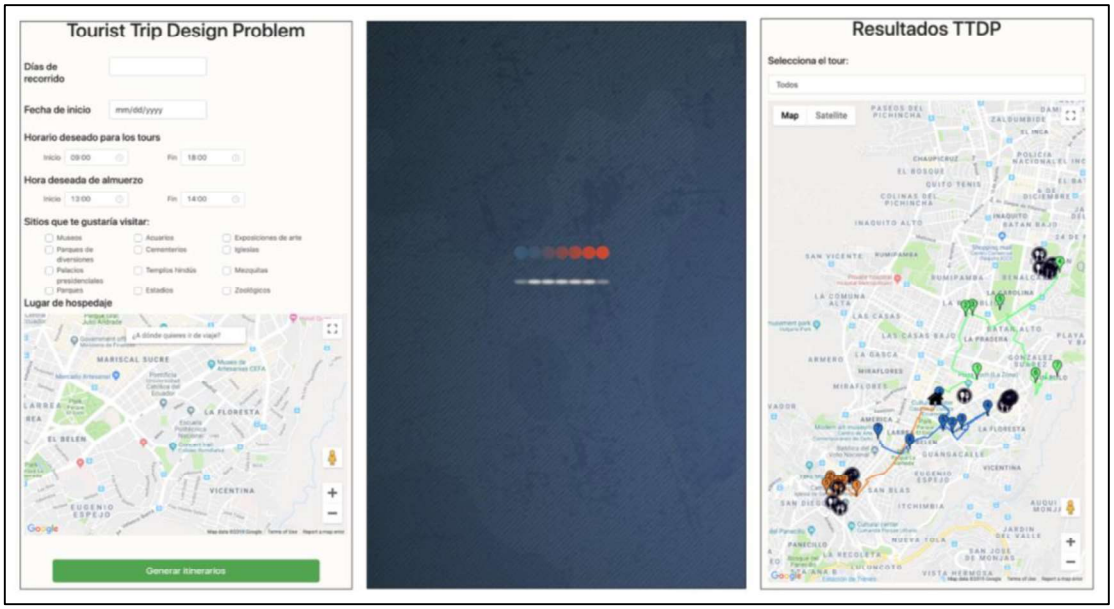

Figura 48. Capturas de pantalla de la aplicación final

3.1.1 Toma de Datos

Se recolectó información que se ajuste con las preferencias del turista. La Figura 49 corresponde al número de días de recorrido y la fecha de inicio de los tours

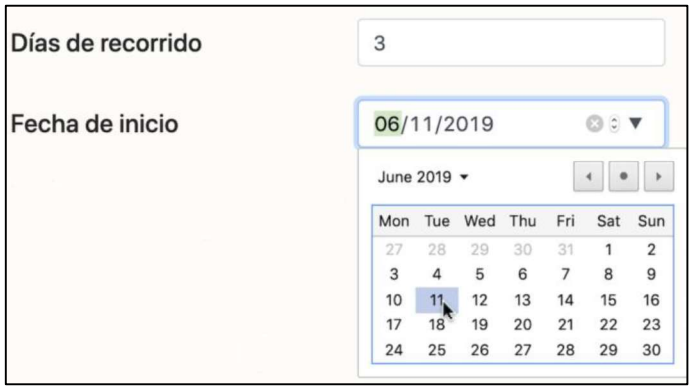

Figura 49. Días de recorrido y fecha de inicio

También se puede fijar el horario para los tours y la hora de almuerzo (Ver Figura 50). De esta manera se puede crear itinerarios más personalizados.

Figura 50. Horarios deseados para tours y el almuerzo

En la Figura 51 se solicita el tipo de lugares turísticos que al turista le gustaría visitar. Como se mencionó anteriormente, los tipos se ajustaron de acuerdo con los ofrecidos por la API de Google.

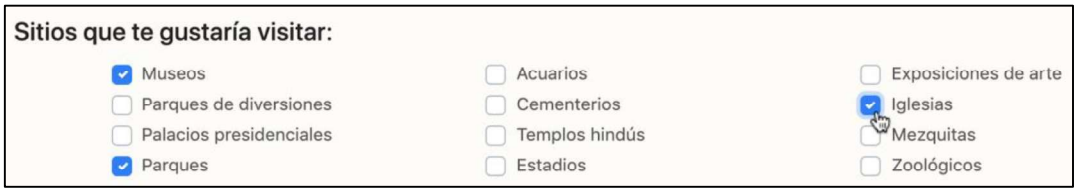

Figura 51. Checklist del de los tipos de puntos de interés

Finalmente, se selecciona el destino mediante un mapa interactivo (ver Figura 52). Con todos estos datos, ya se puede obtener los puntos de interés y generar itinerarios personalizados.

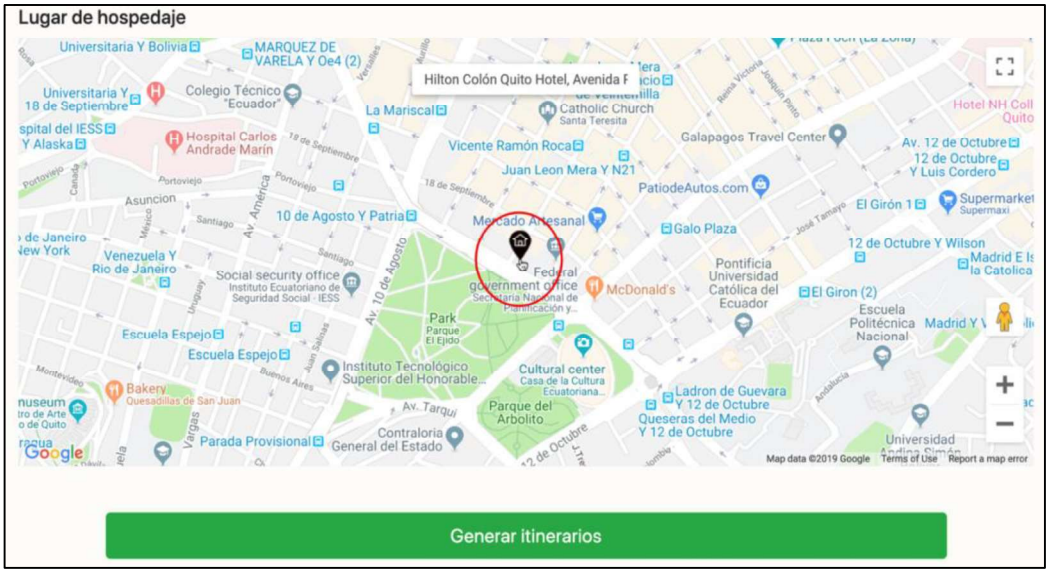

Figura 52. Mapa interactivo para fijar el lugar de hospedaje

3.1.2 Itinerarios Creados

La Figura 53 corresponde al resultado global de todos los itinerarios generados, cada tour se lo representó con un color diferente y se enumeró el orden de visita de los puntos. Se puede notar que el número de clústeres equivale al número de días de visita establecido.

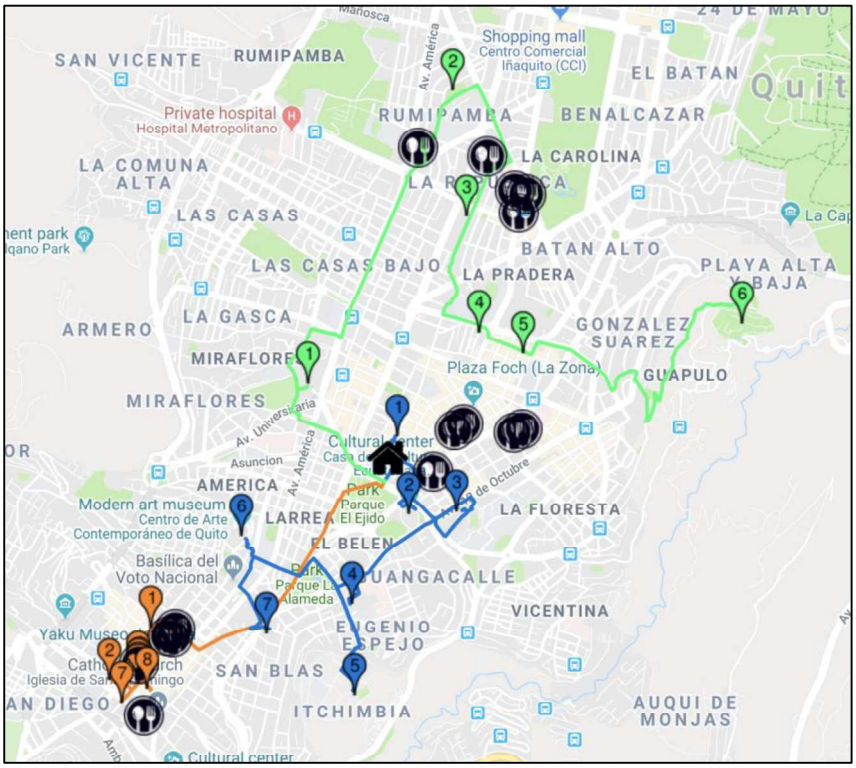

Figura 53. Ejemplo de itinerarios generados

Tras seleccionar el tour de un día en específico (Figura 54), se puede apreciar los detalles puntuales de ese itinerario. En las Figuras 55 y 56 se muestra los detalles correspondientes al segundo día, ya que la fecha de inicio se fijó para el martes 11 de junio del 2019 (ver Figura 49), entonces el segundo corresponde al jueves 12 de junio del 2019.

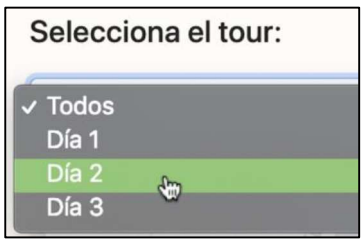

Figura 54. Combo box para seleccionar el tour de un día en específico

Figura 55. Tour del segundo día

Detalles

Horario	Evento
08:55 - 09:00	Salida hacia Santa Teresita
09:00 - 10:20	Visita de Santa Teresita
10:20 - 10:32	Salida hacia Museo Nacional del Ecuador
10:32 - 11:46	Visita de Museo Nacional del Ecuador
11:46 - 11:59	Salida hacia Museo Jacinto Jijón y Caamaño
11:59 - 13:07	Visita de Museo Jacinto Jijón y Caamaño
13:07 - 14:07	Hora de almuerzo
14:07 - 14:26	Salida hacia National Museum of Medicine
14:26 - 15:04	Visita de National Museum of Medicine
15:04 - 15:22	Salida hacia Parque Itchimbía
15:22 - 16:38	Visita de Parque Itchimbía
16:38 - 17:06	Salida hacia Centro de Arte Contemporáneo de Quito
17:06 - 17:46	Visita de Centro de Arte Contemporáneo de Quito
17:46 - 18:00	Salida hacia Iglesia San Blas
18:00 - 18:30	Visita de Iglesia San Blas

Figura 56. Detalles del itinerario del segundo día

En la Figura 57 se observa que aparece una ventana con información adicional cuando se da clic sobre el ícono de un marcador. Además, se aprecia que las recomendaciones de restaurantes se hicieron de acuerdo con la ubicación del turista cuando ha llegado la hora de almuerzo.

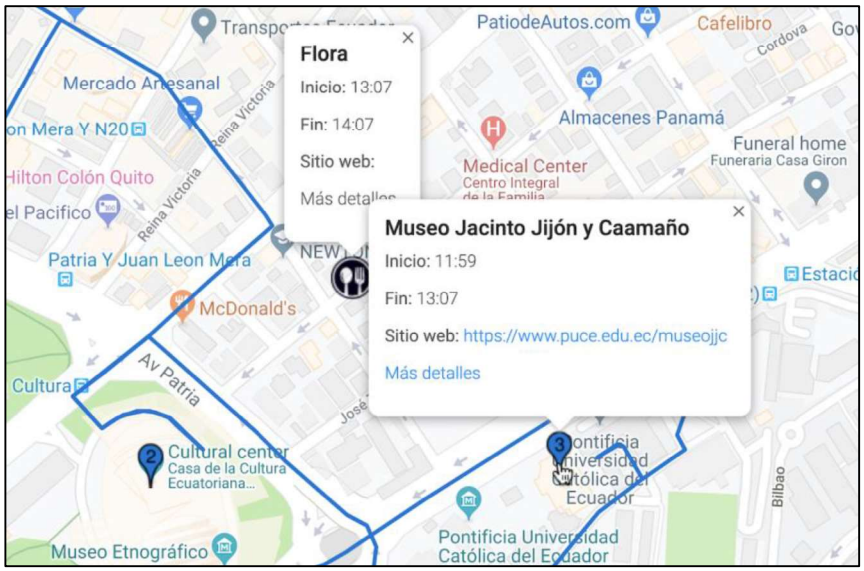

Figura 57. Ventanas de información adicional

3.2 Análisis de itinerarios

En la Figura 58 se representó el resultado del itinerario creado para el segundo día (ver Figura 56) con los horarios de atención de cada sitio (ver Tabla 16). Se puede observar que la optimización de los puntos de interés funciona correctamente, ya que el itinerario se ajusta de acuerdo con los horarios de almuerzo y tours fijados en la Figura 50.

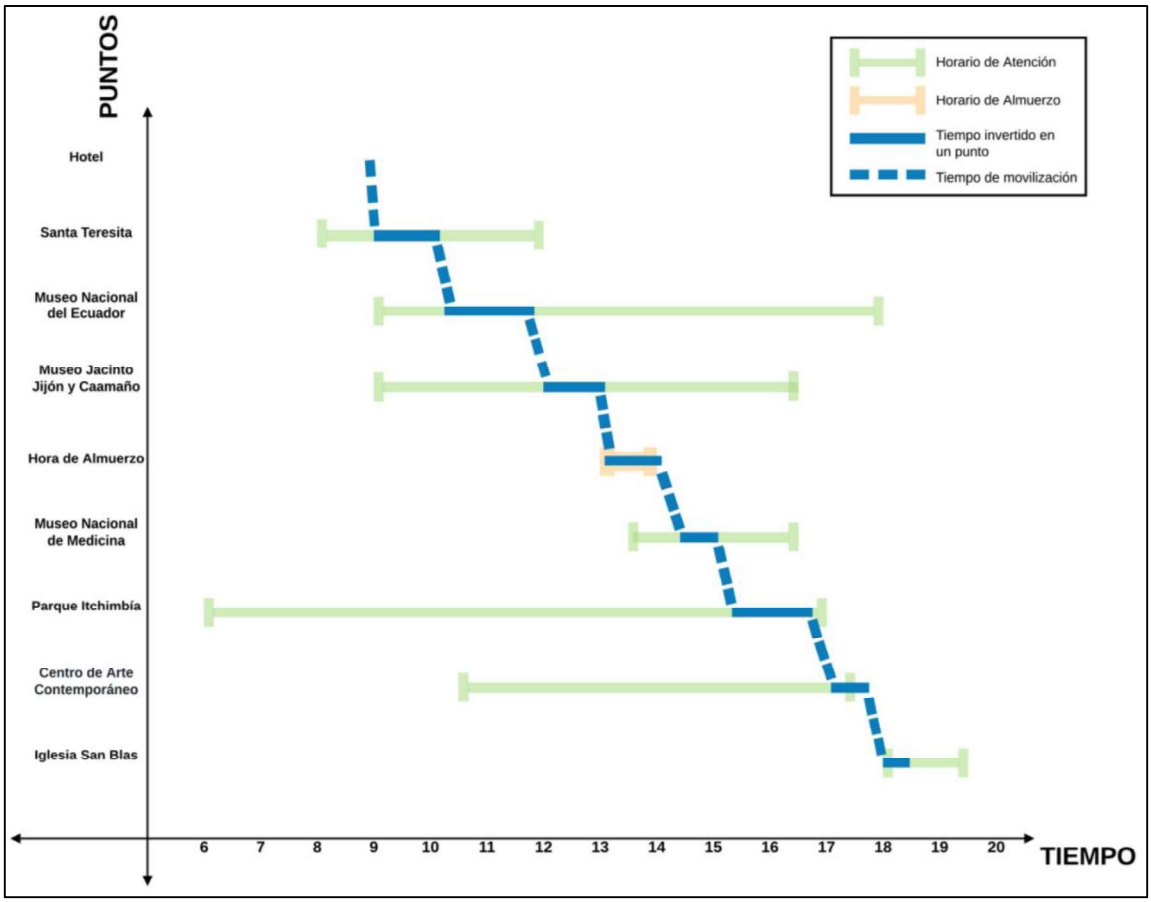

Figura 58. Itinerario generado para el segundo día de tour.

Existen unas pequeñas imperfecciones: la hora del almuerzo empieza 7 minutos más tarde de lo esperado, que es casi insignificante; y la visita al Centro de Arte Contemporáneo (CAC) se recomienda realizarla entre las 17H06 y 17H46, mientras que el lugar sólo atiende entre las 10H30 y 17H30, es decir, hay 16 minutos excedidos. Si se reajusta al itinerario para visitar el CAC dentro de sus horarios de atención podría resultar más costoso, ya que se puede desperdiciar más tiempo en llegar hasta allá desde algún otro punto (ver mapa de la Figura 59) o también perjudicar a más POIs en sus horarios de atención.

Tabla 16. Horarios de atención de los POIs para el jueves 11/06/2019 de acuerdo con la API de Google Maps.

Sitio	Hora de apertura	Hora de cierre
Santa Teresita	08H00	12H00
Museo Nacional del Ecuador	09H00	18H00
Museo Jacinto Jijón y Caamaño	09H00	16H30
Museo Nacional de Medicina	13H30	16H30
Parque Itchimbía	06H00	17H00
Centro de Arte Contemporáneo	10H30	17H30
Iglesia San Blas	18H00	19H30

Los puntos de interés recomendados son de gran calidad ya que se seleccionaron aquellos que atienden durante los días de los tours. Además, en la Figura 59 se visualiza que concuerdan con las preferencias seleccionadas (ver Figura 51), en total se recomendaron 20 puntos para tours de 09H00 a 18H30 durante 3 días.

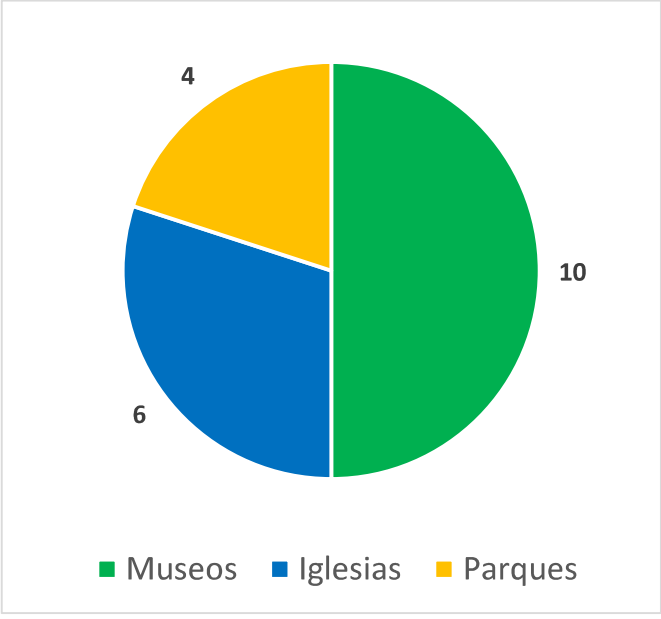

Figura 59. Puntos de interés recomendados para los 3 días de tours.

3.3 Pruebas con usuarios finales

Se realizó pruebas del producto final con un total de 19 personas que trabajan en Kushki, una empresa de comercio electrónico. Donde se evaluó la utilidad y facilidad de uso percibidas para los itinerarios recomendados de acuerdo con el modelo TAM (*Technology Acceptance Model*) [39]. Se efectuaron un total de 6 preguntas por cada usuario, 3 para la utilidad percibida y 3 para la facilidad de uso; la respuesta de cada pregunta se ajustó a una escala del 1 al 7 (donde 1 es la peor calificación y 7 es la mejor). Las preguntas realizadas y sus resultados se encuentran adjuntados en el Anexo VI.

3.3.1 Utilidad percibida

La Figura 60 es un gráfico del promedio de respuesta de cada pregunta con respecto a la utilidad percibida. La 1era pregunta se refiere al ahorro de tiempo que un turista tendría a la hora de planificar itinerarios, y obtuvo el mejor promedio (6.53/7) debido a que los usuarios no tuvieron que invertir mucho tiempo para obtener itinerarios que se acoplaban con el horario que ellos deseaban.

En la pregunta 2 se consultó si las recomendaciones facilitaron la planificación de itinerarios, y también consiguió un promedio alto (6.37/7) debido a que no tuvieron que preocuparse por seleccionar los lugares por visitar, fijarse en sus horarios de atención, etc. simplemente proporcionaron unos pocos datos de entrada.

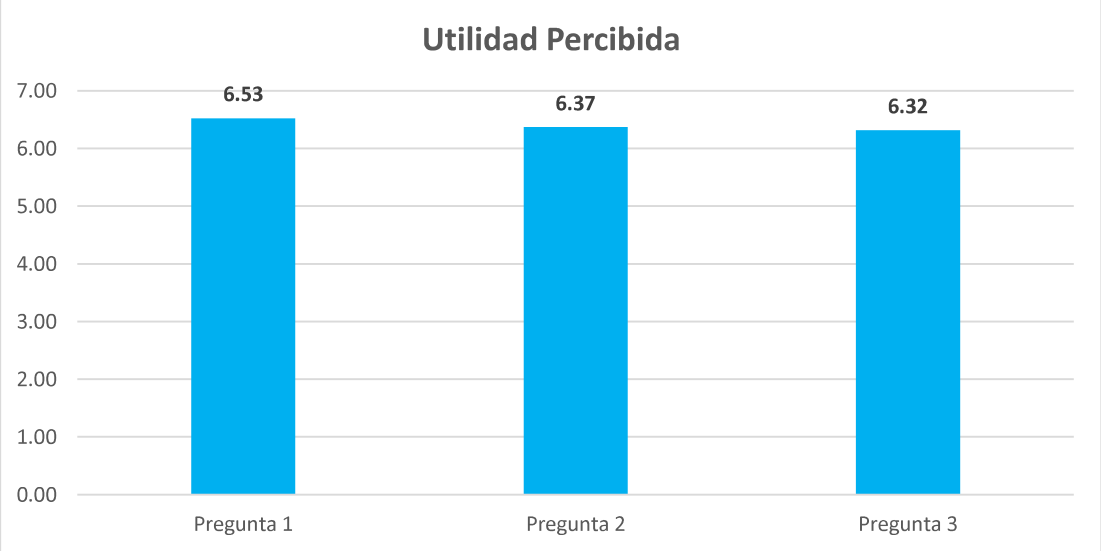

Figura 60. Promedio por pregunta de la utilidad percibida en el producto final

En la última pregunta se consultó si usarían las recomendaciones dentro de sus vacaciones, y de igual manera se obtuvo un buen nivel de aprobación (6.32/7), esto quiere decir que los usuarios sí encontraron útiles a los resultados obtenidos con el algoritmo.

3.3.2 Facilidad de uso percibida

Los promedios de cada pregunta se encuentran en la Figura 61. En la 1era pregunta se consultó si los itinerarios recomendados fueron claros y entendibles y se consiguió un promedio alto de 6.21/7. Esto se debe a que los usuarios podían visualizar el detalle conciso de cada itinerario y determinar si encajaban con sus preferencias proporcionadas.

La pregunta 2 se refiere a la flexibilidad que ofrecían los itinerarios recomendados y se puede visualizar que obtuvo un promedio relativamente bajo de 4.84/7. Puesto que, cuando un itinerario es generado, éste ya no puede ser reajustado en caso de que el usuario no esté conforme con alguna recomendación.

Por último, en la 3era pregunta se consultó sobre la sencillez para obtener los itinerarios. En este caso se consiguió un buen puntaje (6.11/7) porque los usuarios no necesitaban de muchos datos de entrada para obtener las recomendaciones, facilitando cualquier trabajo de planificación.

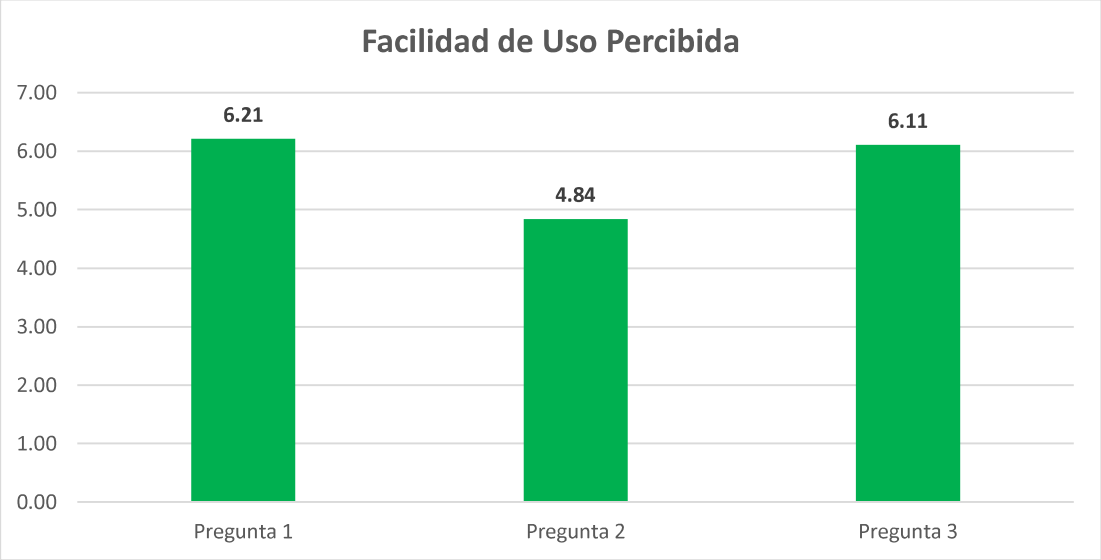

Figura 61. Promedio por pregunta de la facilidad de uso percibida en el producto final

4 CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- El algoritmo diseñado ofrece una solución alterna al problema TTPD. Se trabajó primordialmente con valores de tiempo (fechas de tours, días de visita, horarios de tours, horario de comida, etc.). Por lo cual, la satisfacción del turista mejoraría al disponer de itinerarios optimizados que le permitan conocer más sitios turísticos durante su viaje.
- La agrupación de los puntos de interés por sectores geográficos permite optimizar el tiempo de movilización entre los sitios de un tour ya que la distancia entre ellos es más corta. Sin embargo, podría haber sectores donde sus puntos estén dispersados con una mayor distancia de separación.
- La parametrización de horarios brinda una experiencia más personalizada ya que se ajustan a las inclinaciones del turista. Existen turistas entusiastas y llenos de energía que pueden pasar todo el día visitando distintos puntos de la ciudad; pero también hay otros que dedican más tiempo a descansar y no están tan interesados en realizar largos recorridos.
- Los puntos de interés varían de acuerdo con la ubicación del destino del turista, no todas las ciudades disponen de la misma diversidad turística. En ciudades andinas, no se podría encontrar acuarios o parques acuáticos; no todos los destinos contarían con parques de diversiones, mezquitas, zoológicos, etc.
- El tiempo estimado de visita para un lugar turístico es una métrica ambigua, no existe ninguna API que proporcione esa información. Esto se debe a que varía significantemente de acuerdo con la voluntad y disponibilidad de tiempo de un turista. Por tal razón, se fijó un tiempo aleatorio diferente para cada lugar que se ajusta a un tiempo promedio de visita (entre 40 minutos y 2 horas).
- El tiempo estimado de viaje entre dos lugares varía extensamente de acuerdo con el medio de transporte utilizado. Si se moviliza en algún transporte motorizado, el tiempo depende del tráfico de la ciudad y las horas pico, cosa que podría perjudicar notablemente al momento de optimizar un tour. Por tal motivo, se utilizó el tiempo promedio que se tarda movilizándose a pie, ya que se tiene un pronóstico más acertado y la distancia que separa a los puntos de interés es relativamente corta por estar agrupados geográficamente.
- La calidad de los itinerarios depende de los puntos de interés obtenidos dentro de las fechas fijadas por el turista. Podrían haber días donde pocos sitios estén

disponibles para visitar o también surgir conflictos entre los diferentes horarios de atención de los sitios.

- La API de Google Maps provee información importante en el sector turístico. Sin embargo, el algoritmo puede funcionar con cualquier otra API de los mismos fines, sólo basta proporcionar los mismos datos de entrada. Tras parametrizar aquellos datos, se asegura independencia de recursos ya que en algún futuro puede aparecer algún otro servicio que sea más útil y se lo reemplazaría sin mucho esfuerzo.
- Las recomendaciones de restaurantes se hicieron de acuerdo con la posición geográfica del turista cuando se acerca la hora del almuerzo. Puede haber casos en donde haya pocos puntos de interés y jamás se llegue a cruzar con el horario de comida, por lo que no aparecerían dichas recomendaciones.

4.2 Recomendaciones

Para trabajos futuros se recomienda:

- Utilizar una medida más real para el tiempo estimado de visita para cada lugar turístico ya que actualmente no existe ningún API que proporcione tal información.
- Ofrecer la flexibilidad de reajustar los itinerarios en caso de que los turistas no estén de acuerdo con algún detalle dentro de las recomendaciones.
- Usar algún API de clima, ya que algunos sitios turísticos se encuentran al aire libre y mediante una predicción de clima, se podría saber si es aconsejable visitarlos.
- Recomendar sitios que se ajusten a las preferencias de comida del turista ya que la dieta podría variar de acuerdo con cada persona (vegetarianos, veganos, etc.).
- En cuanto a movilidad, tomar en cuenta cuáles serían los medios de transporte más adecuados durante la movilización entre los diferentes puntos. De este modo, se crearían itinerarios más detallados y con un tiempo más optimizado.
- Incluir un presupuesto de dinero para crear recomendaciones que se ajusten al bolsillo del turista, ya que algunos sitios podrían resultar sumamente costosos visitarlos.

5 REFERENCIAS BIBLIOGRÁFICAS

[1] A. Duarte Muñoz, J. J. Pantrigo Fernández y M. Gallego Carrillo, Metaheurísticas, Madrid: Madrid Dykinson, 2007.

[2] O. Suarez, «Una aproximación a la heurística y metaheuristicas,» *INGE@ UAN-Tendencias en la Ingeniería,* vol. 1, nº 2, 2013.

[3] H. Ding, L. Ke y Z. Geng, «Route planning in a new tourist recommender system: A fireworks algorithm based approach,» de *IEEE Congress on Evolutionary Computation (CEC)*, Vancouver, BC, Canada, 2016.

[4] J. Brito, A. Expíósito-Márquez y J. A. Moreno, «A fuzzy GRASP algorithm for solving a Tourist Trip Design Problem,» de *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Naples, Italy, 2017.

[5] D. F. Valencia Vargas y D. A. Poveda Sendales, «Optimal scheduling trip plan for tourist,» de *10th International Conference on Intelligent Systems and Control (ISCO)*, Coimbatore, India, 2016.

[6] W. Zheng, Z. Liao y J. Qin, «Using a four-step heuristic algorithm to design personalized day tour route within a tourist attraction,» *Tourism Management,* vol. 62, pp. 335-349 , 2017.

[7] N. Ganganath, C.-T. Cheng y C. K. Tse, «Data Clustering with Cluster Size Constraints Using a Modified K-Means Algorithm,» de *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, Shanghai, China, 2014.

[8] J.-Y. Potvin, «Genetic algorithms for the traveling salesman problem,» *Annals of Operations Research,* vol. 63, nº 3, p. 37–370, 1996.

[9] R. Guerequeta y A. Vallecillo , Técnicas de Diseño de Algoritmos, Servicio de Publicaciones de la Universidad de Málaga, 1998.

[10] O. Goldreich, P, NP, and NP-Completeness: The Basics of Computational Complexity, Cambridge University Press, 2010.

[11] R. K. Arora, Optimization: Algorithms and Applications, CRC Press, 2015.

[12] R. B. Navarro, Meta-Heurísticas híbridas para optimización mono-objetivo y multi-objetivo. Paralelización y aplicaciones, Universidad Almería, 2008.

[13] D. Gavalas, C. Konstantopoulos, K. Mastakas y G. Pantziou, «A survey on algorithmic approaches for solving tourist trip design problems,» *Journal of Heuristics,* vol. 20, nº 3, p. 291–328, 2014.

[14] «Metaheurísticas: Una visión global,» *Inteligencia artificial: Revista Iberoamericana de Inteligencia Artificial,* vol. VII, nº 9, pp. 7-28, 2003.

[15] Universidad de le República Uruguay, «Curso: Algoritmos Evolutivos,» [En línea]. Available: https://eva.fing.edu.uy/course/view.php?id=1049. [Último acceso: 5 February 2019].

[16] M. Gestal, D. Rivero, J. R. Rabuñal, J. Dorado y A. Pazos, Introducción a los algoritmos genéticos y a la programación genética, Universidade da Coruña, Servizo de Publicacións, 2010.

[17] S. Sivanandam y S. N. Deepa, Introduction to Genetic Algorithms, Springer-Verlag Berlin Heidelberg, 2008.

[18] D. B. Fogel, «What is evolutionary computation?,» *IEEE Spectrum,* vol. 37, nº 2, pp. 26-32, 2000.

[19] M. Gen y R. Cheng, Genetic Algorithms and Engineering Optimization, John Wiley & Sons, 2000.

[20] S. Sumathi, T. Hamsapriya y P. Surekha, Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab, Springer Science & Business Media, 2008.

[21] P. Fleming y A. Zalzala, Genetic Algorithms in Engineering Systems, IET, 1997.

[22] O. Kramer, Genetic Algorithm Essentials, Springer International Publishing, 2017.

[23] M. Obitko, «Introduction to Genetic Algorithms - Tutorial with Interactive Java Applets,» [En línea]. Available: https://www.obitko.com/tutorials/genetic-algorithms/parameters.php.

[24] StrongLoop, IBM, and other expressjs.com contributors, «Express,» 2017. [En línea]. Available: https://expressjs.com/.

[25] Google Developers, «Places API,» 2019. [En línea]. Available: https://developers.google.com/places/web-service/intro.

[26] Google Developers, «Directions API,» 2019. [En línea]. Available: https://developers.google.com/maps/documentation/directions/start.

[27] A. Bank, Learning React: Functional Web Development with React and Redux, O'Reilly Media; 1 edition, 2017.

[28] Kinsta, «What Is GitHub? A Beginner's Introduction to GitHub,» 2019. [En línea]. Available: https://kinsta.com/knowledgebase/what-is-github/.

[29] Jetbrains, «WebStorm: The Smartest JavaScript IDE,» 2019. [En línea]. Available: https://www.jetbrains.com/webstorm/.

[30] Momentjs.com, «Moment.js | Home,» 2019. [En línea]. Available: https://momentjs.com/.

[31] Mochajs.org, «Mocha - the fun, simple, flexible JavaScript test framework,» 2019. [En línea]. Available: https://mochajs.org/.

[32] Chaijs.com, «Chai Assertion Library,» 2019. [En línea]. Available: https://www.chaijs.com/.

[33] K. S. Rubin, Essential Scrum: A Practical Guide to the Most Popular Agile Process, Addison-Wesley, 2012.

[34] J. Sutherland y K. Schwaber, «Scrum Guides,» November 2017. [En línea]. Available: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Spanish-SouthAmerican.pdf.

[35] J. Schiel, The ScrumMaster Study Guide, CRC Press, 2016.

[36] Google Maps Platform, «Google Developers,» [En línea]. Available: https://developers.google.com/places/web-service/search.

[37] Google Maps Platform, [En línea]. Available: https://developers.google.com/places/web-service/supported_types.

[38] hapijs, «Github,» [En línea]. Available: https://github.com/hapijs/joi/blob/v15.0.0/API.md.

[39] F. D. Davis, «Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology,» *MIS Quarterly,* vol. 13, pp. 319-340, 1989.

6 ANEXOS

6.1 Anexo I: Historias Épicas

Tabla 17. Historia Épica AH1

HISTORIA DE USUARIO ÉPICA	
ID: AH1	**Prioridad:** Alta
Título: Integración del algoritmo de *clustering* con la API de Google Maps.	
Descripción: Como turista, necesito extraer puntos de mi interés agrupados en sectores geográficos.	

Tabla 18. Historia Épica AH2

HISTORIA DE USUARIO ÉPICA	
ID: AH2	**Prioridad:** Alta
Título: Desarrollo del algoritmo genético personalizado.	
Descripción: Como turista, necesito obtener itinerarios optimizados para cada grupo geográfico de puntos de interés.	

Tabla 19. Historia Épica AH3

HISTORIA DE USUARIO ÉPICA	
ID: AH3	**Prioridad:** Media
Título: Desarrollo de la interfaz para el usuario final.	
Descripción: Como turista, necesito de una interfaz amigable, de manera que pueda visualizar mis itinerarios turísticos generados de acuerdo con mis preferencias.	

6.2 Anexo II: Historias de Usuario del Sprint 1

Tabla 20. Historia de Usuario AH1-01

HISTORIA DE USUARIO		
ID: AH1-01	**Días estimados:** 5	
Título: Extracción de los puntos de interés del turista.		
Sprint N°: 1	**Prioridad:** Alta	**Estado:** Por implementar
Descripción: Como turista, deseo visitar lugares turísticos de mi preferencia que se encuentren cerca del lugar de hospedaje.		
Criterios de aceptación	El servicio deberá extraer sitios turísticos cercanos a la ubicación de destino del turista.	
	Los sitios turísticos se deben ajustar a los gustos del turista.	

Tabla 21. Historia de Usuario AH1-02

HISTORIA DE USUARIO		
ID: AH1-02	**Días estimados:** 9	
Título: Agrupación geográfica de los POIs.		
Sprint N°: 1	**Prioridad:** Alta	**Estado:** Por implementar
Descripción: Como turista, deseo que los sitios turísticos por visitar estén relativamente cerca para cada día de recorrido, de manera que pueda movilizarme a pie sin ningún problema.		
Criterios de aceptación	Dado un conjunto de puntos de interés, cuando se los agrupe geográficamente, las distancias entre los puntos de cada sector deben ser relativamente cortas.	
	El número de sectores geográficos debe equivaler al número de días de visita del turista.	

6.3 Anexo III: Historias de Usuario del Sprint 2

Tabla 22. Historia de Usuario AH2-01

HISTORIA DE USUARIO		
ID: AH2-01	**Días estimados:** 13	
Título: Generación de itinerarios turísticos optimizados.		
Sprint N°: 2	**Prioridad:** Alta	**Estado:** Por implementar
Descripción: Como turista, deseo saber el orden de visita de mis puntos de interés, de manera que pueda optimizar el tiempo de recorrido.		
Criterios de aceptación	Dado un grupo geográfico de puntos turísticos, cuando se genere un itinerario óptimo, se deberá saber el orden de cómo visitarlos.	
	El itinerario debe tener en cuenta el horario de almuerzo entre la 1pm y 2pm de la tarde.	
	El itinerario debe considerar los horarios de atención y el tiempo de movilización entre los diferentes puntos.	
	El itinerario debe tener un tiempo de recorrido de aproximadamente 8 horas.	

Tabla 23. Historia de Usuario AH2-02

HISTORIA DE USUARIO		
ID: AH2-02	**Días estimados:** 1	
Título: Horarios de atención y tiempos de viaje de los POIs.		
Sprint N°: 2	**Prioridad:** Media	**Estado:** Por implementar
Descripción: Como turista, deseo conocer los horarios de atención de cada lugar turístico y el tiempo de viaje para la movilización entre ellos.		
Criterios de aceptación	Cada punto de interés debe contener información sobre sus horarios de atención.	
	Se debe descartar los sitios que no atiendan durante los días de visita del turista.	
	Se debe conocer el tiempo de movilización (a pie) entre los puntos de un sector geográfico.	

6.4 Anexo IV: Historias de Usuario del Sprint 3

Tabla 24. Historia de Usuario AH2-01 actualizada para el último Sprint

HISTORIA DE USUARIO		
ID: AH2-01	**Días estimados:** 2	
Título: Generación de itinerarios turísticos optimizados.		
Sprint N°: 3	**Prioridad:** Alta	**Estado:** En desarrollo
Descripción: Como turista, deseo saber el orden de visita de mis puntos de interés, de manera que pueda optimizar el tiempo de recorrido.		
Criterios de aceptación	Dado un grupo geográfico de puntos turísticos, cuando se genere un itinerario óptimo, se deberá saber el orden de cómo visitarlos.	
	El itinerario debe tener en cuenta el horario de almuerzo fijado por el turista.	
	El itinerario debe considerar los horarios de atención y el tiempo de movilización entre los diferentes puntos.	
	El itinerario debe tener un tiempo de recorrido establecido por el turista.	

Tabla 25. Historia de Usuario AH2-03

HISTORIA DE USUARIO		
ID: AH2-03	**Días estimados:** 2	
Título: Parametrización de los horarios de tour y de almuerzo.		
Sprint N°: 3	**Prioridad:** Alta	**Estado:** Por implementar
Descripción: Como turista, deseo establecer los horarios de tours y de almuerzo, con el fin de gozar una experiencia más personalizada.		
Criterios de aceptación	Los itinerarios generados deberán acoplarse con los horarios establecidos por el turista.	

Tabla 26. Historia de Usuario AH2-04

HISTORIA DE USUARIO		
ID: AH2-04	**Días estimados:** 1	
Título: Recomendaciones de restaurantes para el horario de comida.		
Sprint N°: 3	**Prioridad:** Media	**Estado:** Por implementar
Descripción: Como turista, deseo tener recomendaciones de restaurantes cuando se acerque la hora del almuerzo.		
Criterios de aceptación	Los restaurantes recomendados deben estar cerca de la posición del turista cuando caiga la hora de almuerzo.	

Tabla 27. Historia de Usuario AH3-01

HISTORIA DE USUARIO		
ID: AH3-01	**Días estimados:** 5	
Título: Desarrollo del formulario del usuario final.		
Sprint N°: 3	**Prioridad:** Media	**Estado:** Por implementar
Descripción: Como turista, deseo proporcionar los datos de mis preferencias en un formulario sencillo antes de generar los itinerarios.		
Criterios de aceptación	El formulario debe recopilar información como días de recorrido, fecha de inicio de viaje, horarios deseados para tours y almuerzo, preferencias turísticas y lugar de hospedaje.	
	El lugar de hospedaje debe ser seleccionado en un mapa interactivo.	
	Cuando el turista envíe la información completa de sus preferencias, entonces debería aparecer una pantalla de carga.	
	Cuando el turista lo intente enviar su información incompleta, se deberán mostrar los campos que faltan por llenar.	

Tabla 28. Historia de Usuario AH3-02

HISTORIA DE USUARIO		
ID: AH3-02	**Días estimados:** 4	
Título: Desarrollo de la interfaz de despliegue de resultados.		
Sprint N°: 3	**Prioridad:** Media	**Estado:** Por implementar
Descripción: Como turista, deseo visualizar los itinerarios recomendados en una interfaz amigable.		
Criterios de aceptación	La pantalla deberá desplegar los itinerarios en un mapa y trazar las rutas entre los diferentes POIs.	
	Cuando se de clic sobre un POI, se deberá abrir un cuadro con información detallada	
	El turista deberá poder seleccionar el tour de un día en específico.	
	Para cada día, se deberá mostrar un cuadro con la información detallada de todo el itinerario.	

6.5 Anexo V: Diseño de las interfaces de usuario final

Formulario

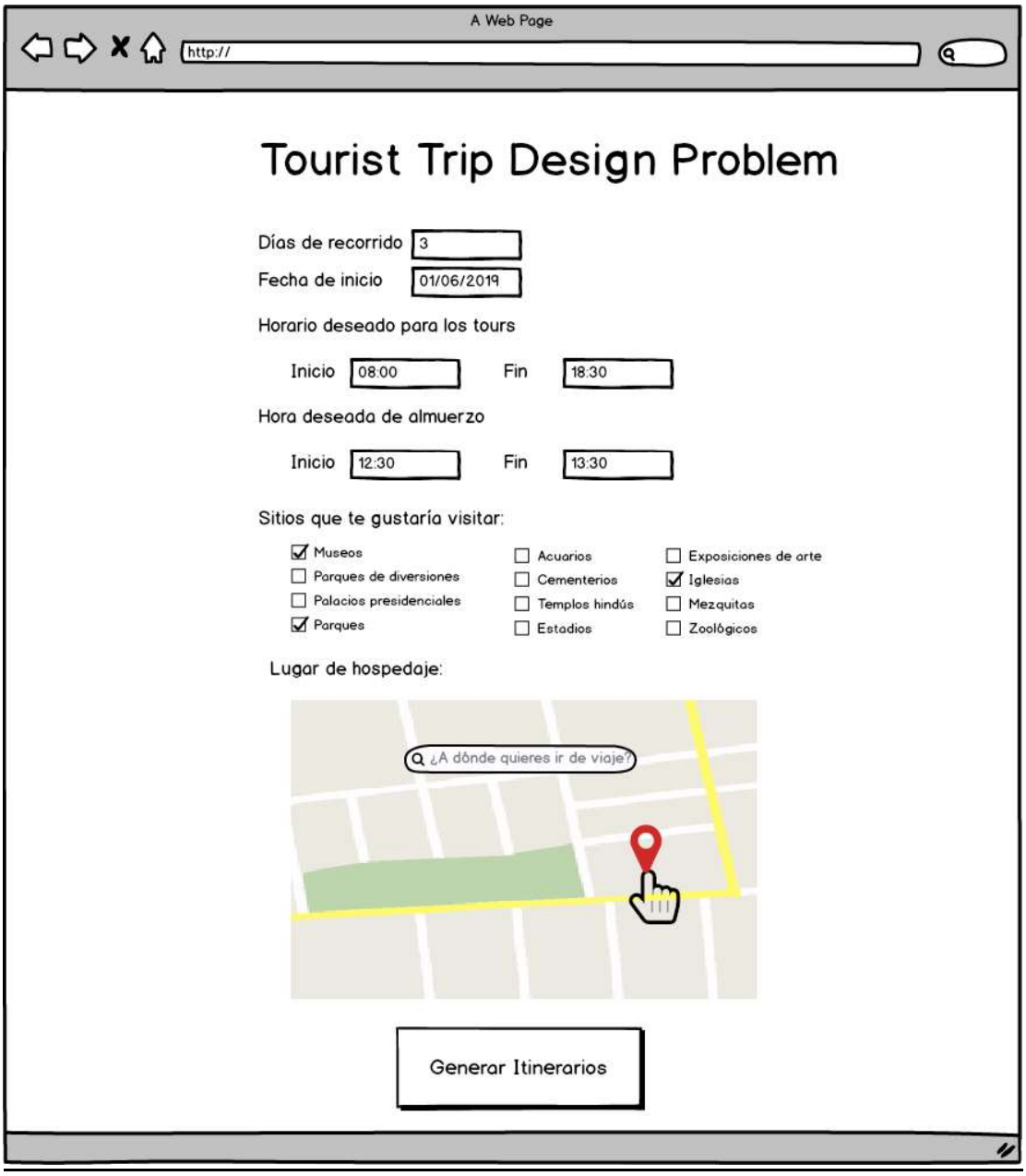

Figura 62. Diseño del formulario para la toma de datos (Realizado en Balsamiq)

Pantalla de resultados

Figura 63. Diseño de la pantalla de resultados (Realizado en Balsamiq)

6.6 Anexo VI: Encuestas Realizadas

Cuestionario de utilidad percibida

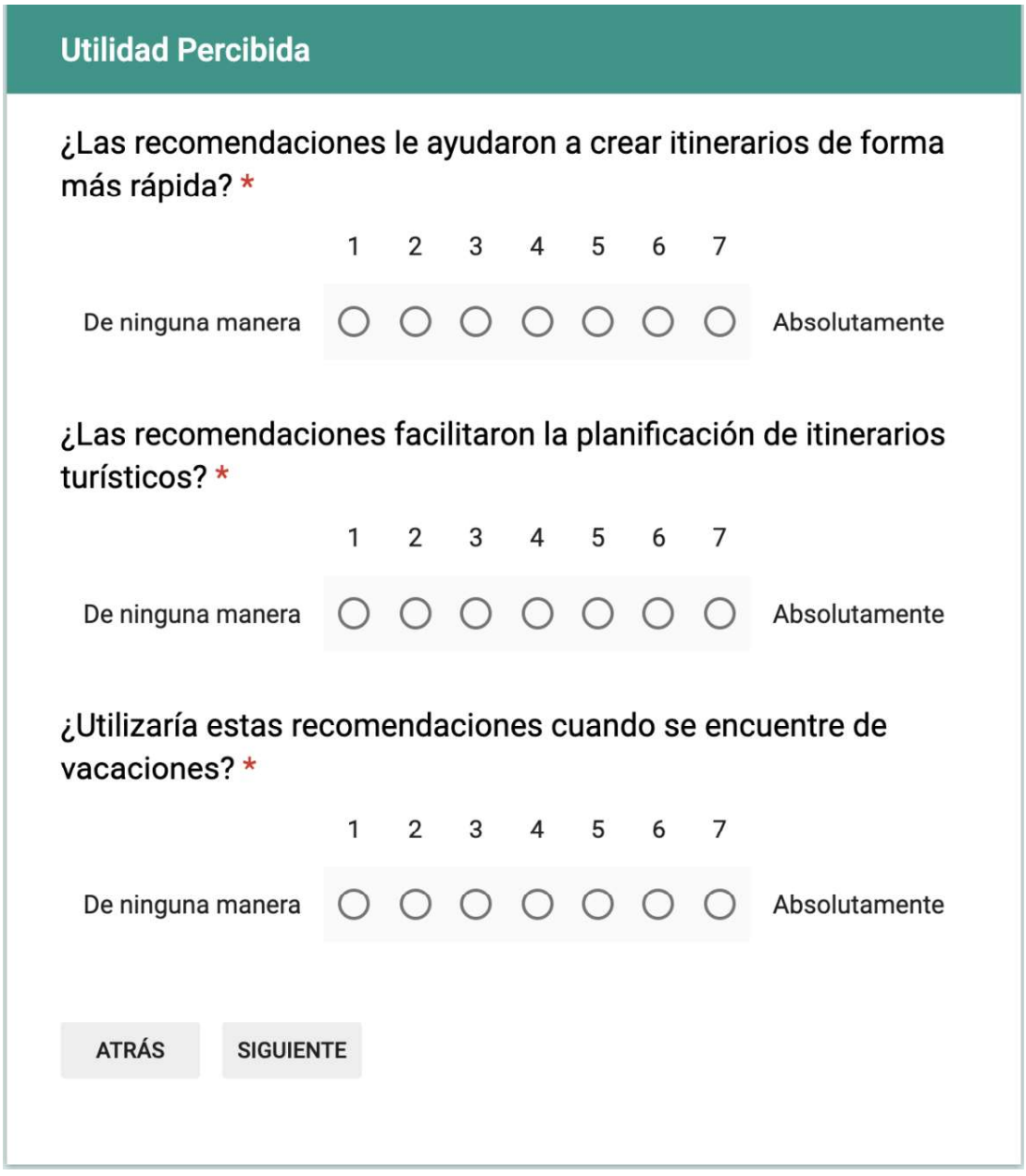

Figura 64. Formulario para la utilidad percibida

Cuestionario de facilidad de uso percibida

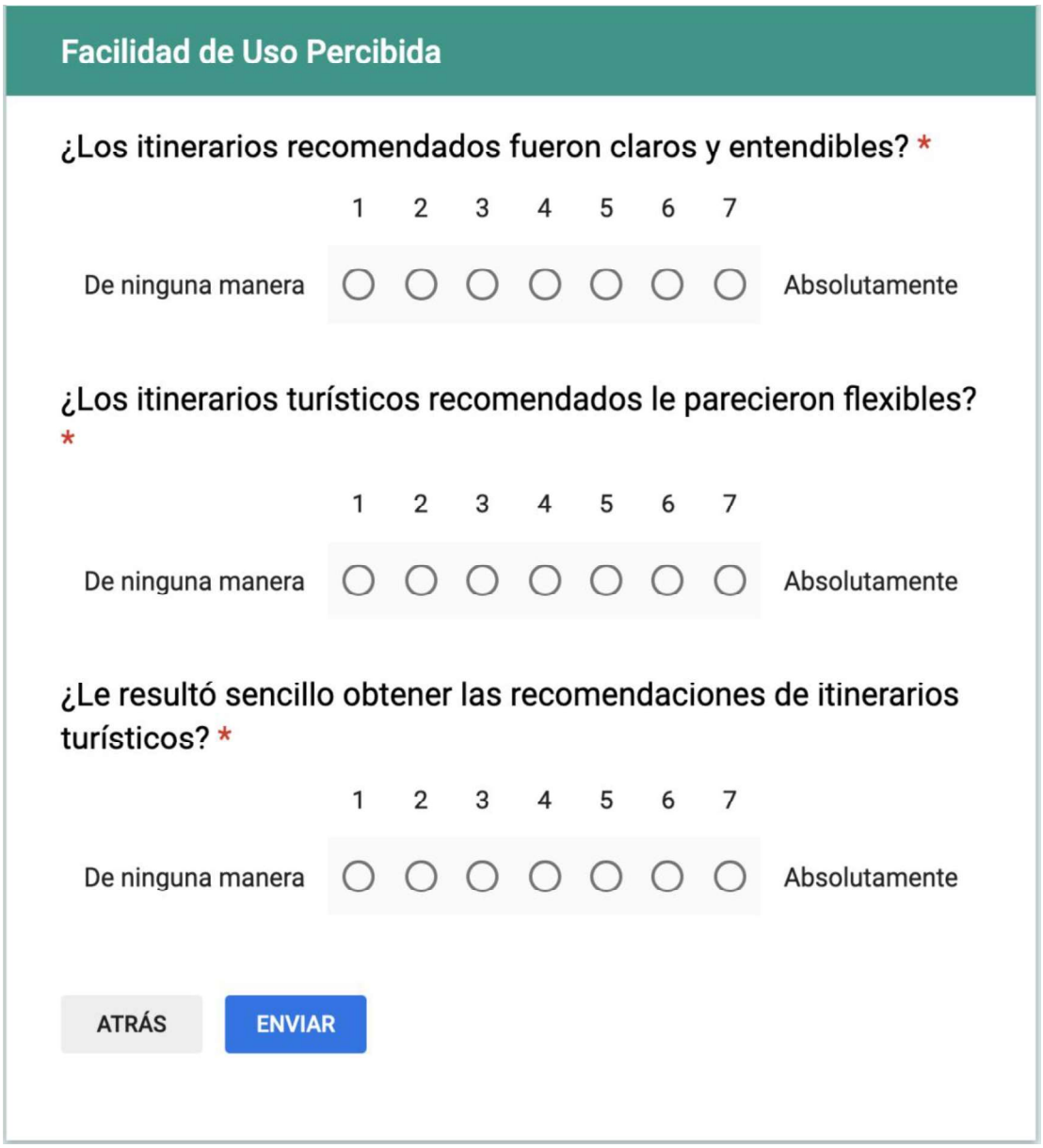

Figura 65. Formulario para la facilidad de uso percibida

Resultados de las encuestas

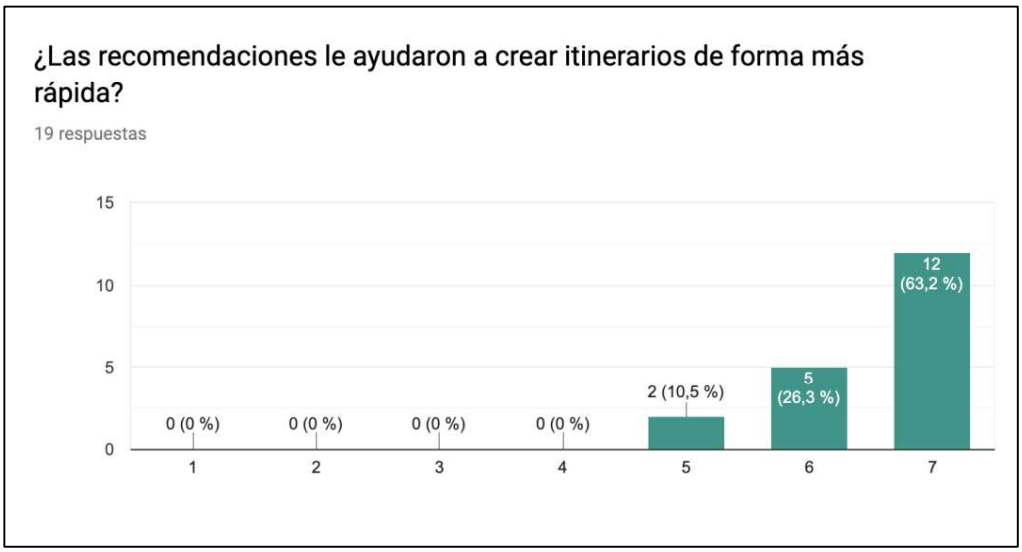

Figura 66. Respuestas de la pregunta 1 para la utilidad percibida

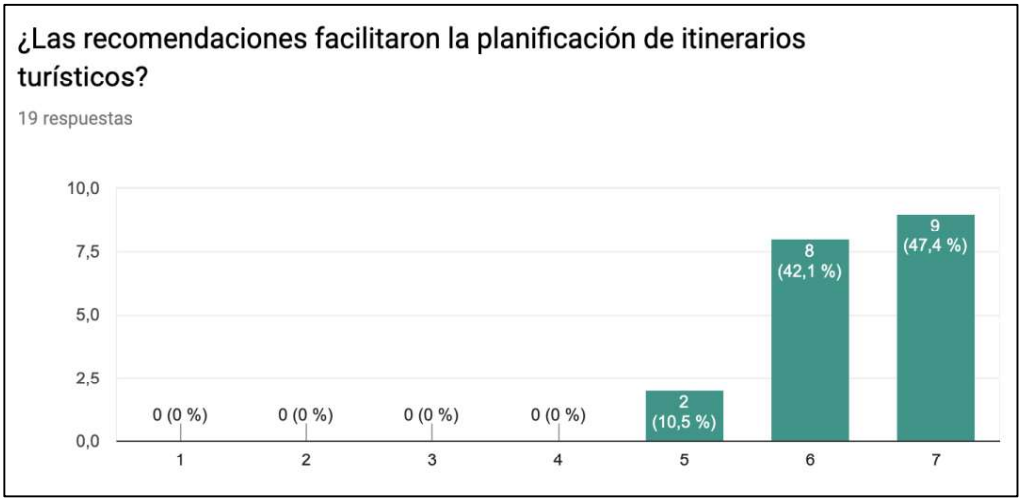

Figura 67. Respuestas de la pregunta 2 para la utilidad percibida

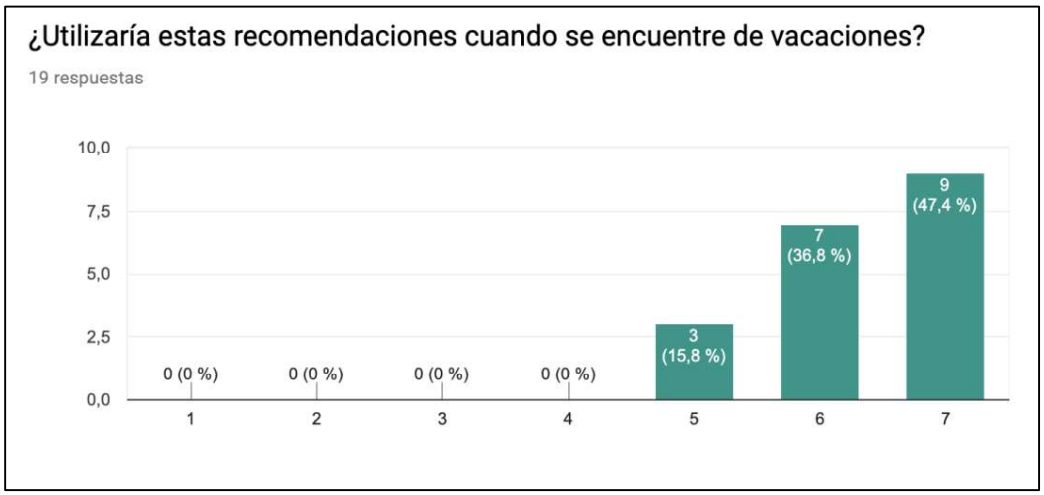

Figura 68. Respuestas de la pregunta 3 para la utilidad percibida

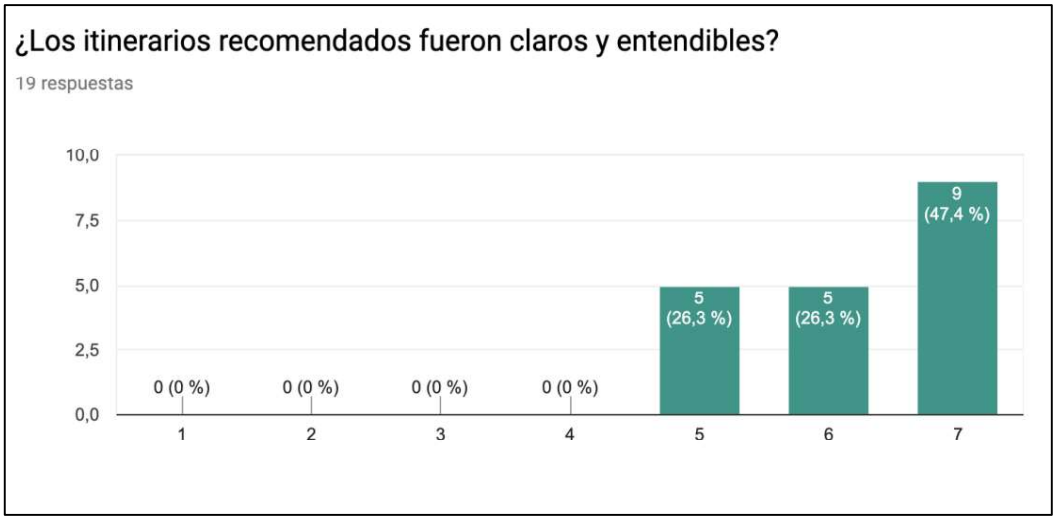

Figura 69. Respuestas de la pregunta 1 para la utilidad de uso percibida

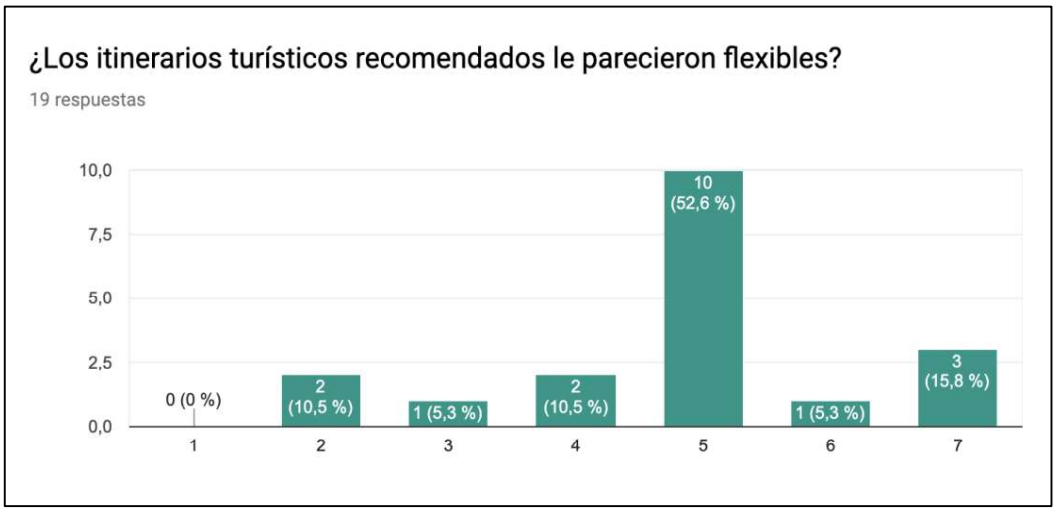

Figura 70. Respuestas de la pregunta 2 para la utilidad de uso percibida

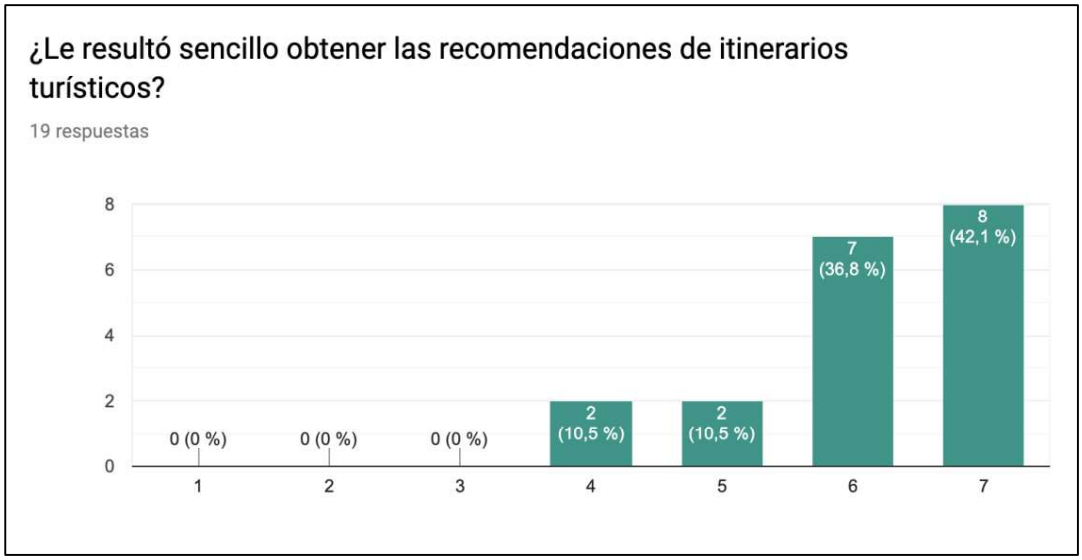

Figura 71. Respuestas de la pregunta 3 para la utilidad de uso percibida