Resolución en línea del problema de viajes compartidos en taxis usando algoritmos evolutivos

Gabriel Fagúndez, Renzo Massobrio y Sergio Nesmachnow
Centro de Cálculo, Facultad de Ingeniería, Universidad de la República
Herrera y Reissig 565, 11300 Montevideo, Uruguay
Email: gabriel.fa07@gmail.com, renzomassobrio@gmail.com, sergion@fing.edu.uy

***Resumen*—Este artículo presenta la aplicación de un algoritmo evolutivo para resolver el problema de distribución de pasajeros que parten de un mismo origen en diferentes taxis, con el objetivo de minimizar el costo total de los viajes. El análisis experimental compara la calidad de soluciones obtenidas por el algoritmo propuesto con una heurística ávida intuitiva para resolver instancias reales del problema, mostrando que el algoritmo evolutivo puede alcanzar significativas mejoras en el costo total de los viajes, de hasta 41.7 % en el mejor caso, y 36.4 % en el caso promedio. Asimismo, se presentan dos aplicaciones desarrolladas para resolver interactivamente (en línea) instancias reales del problema: una interfaz de usuario web y una aplicación móvil para dispositivos con sistema operativo iOS.**

***Abstract*—This article presents the application of evolutionary algorithms for solving the problem of distributing a group of passengers travelling from the same origin to different destinations in several taxis, with the goal of minimizing the total cost of the trips. The experimental analysis compares the quality of the solutions found using the proposed algorithm versus those computed using an intuitive greedy heuristic to solve the problem, showing that the evolutionary algorithm is able to reach significant improvements in the trips' total cost, outperforming the greedy heuristic in up to 41.7 % in the best case, and up to 36.4 % on average. Furthermore, we introduce two applications developed to solve interactively (on-line) realistic problem instances: a web-based user interface and a mobile application for devices using the iOS operating system.**

I. Introducción

Los viajes compartidos en automóvil, conocidos bajo el término anglosajón de *car pooling*, han tenido gran interés público en los últimos años [1]. Los beneficios de compartir transporte afectan tanto en el plano económico como en el ecológico, a niveles individuales y colectivos. Compartir vehículos entre personas que deben transportarse a destinos cercanos permite minimizar los costos de traslado y también la cantidad de automóviles en las rutas, reduciendo la polución y contribuyendo a minimizar el impacto ambiental del transporte, que es un problema capital en las últimas décadas [2], [3] en las grandes ciudades. Adicionalmente, un menor tráfico se traduce directamente en una menor cantidad de embotellamientos, logrando un tránsito más fluido y por ende mayor eficiencia en los traslados, en especial en horas pico. Desde el punto de vista económico, al compartir el costo del viaje entre los diferentes pasajeros del vehículo, se reduce significativamente los costos asociados. En el caso particular de usuarios que utilizan automóviles con taxímetro, los viajes son más rápidos y por lo tanto más baratos.

Los beneficios de los viajes compartidos han dado lugar a diferentes iniciativas para atender el gran interés del público. Entre ellas se pueden mencionar los carriles exclusivos para viajes compartidos presentes en muchas ciudades, las campañas para compartir los viajes hacia el lugar del trabajo, y una gran variedad de aplicaciones en línea desarrolladas para encontrar compañeros de viaje.

Los automóviles con taxímetro (taxis) constituyen un medio de transporte rápido y confortable. Sin embargo, es frecuente que los taxis no se utilicen a capacidad completa. Por este motivo, los taxis podrían verse beneficiados por los aspectos aplicables a los viajes compartidos. En la actualidad, existen algunos servicios web que proporcionan soluciones al problema de planificación de viajes compartidos, y algunos trabajos de investigación que han resuelto variantes del problema de planificar viajes en taxis utilizando métodos específicos.

El problema de planificación de viajes compartidos es NP-difícil [4]. Para resolver instancias con dimensiones realistas es necesario utilizar heurísticas o metaheurísticas [5] que permitan calcular soluciones de calidad aceptable en tiempos razonables.

En esta línea de trabajo, este artículo presenta una solución al problema de planificación de viajes compartidos en taxi, donde los usuarios parten desde un mismo origen hacia varios destinos. Se describe un algoritmo evolutivo eficiente que busca una solución que minimice el gasto total de los pasajeros. La validación experimental del algoritmo propuesto se realiza sobre escenarios reales, incluyendo datos actualizados de ubicaciones geográficas y tarifas de taxímetro. Complementariamente, se presenta un sistema que involucra dos aplicaciones para que los usuarios puedan acceder a la resolución de casos reales del problema en línea: una aplicación web y una aplicación móvil para dispositivos con sistema operativo iOS.

El artículo se estructura del modo que se describe a continuación. La siguiente sección presenta el problema de planificación de recorridos compartidos en taxis y su formulación matemática. Una reseña de los principales trabajos relacionados se presenta en la Sección III. La Sección IV introduce a los algoritmos evolutivos, y la Sección V ofrece los detalles de implementación del algoritmo evolutivo propuesto. El análisis experimental se reporta en la Sección VI, incluyendo la comparación con heurísticas tradicionales para la resolución del problema y un análisis de tiempo de ejecución. Las aplicaciones de software desarrolladas (interfaz web y una aplicación móvil) se describen en la Sección VII. Finalmente, la Sección VIII presenta las conclusiones de la investigación y las principales líneas de trabajo futuro.

II. El problema de viajes compartidos en taxis

Esta sección presenta la descripción del problema de viajes compartidos en taxis y su formulación matemática como problema de optimización combinatoria.

II-A. Descripción y modelo del problema

El problema a resolver modela la siguiente situación: un determinado número de personas, ubicadas en un mismo lugar de origen, deciden viajar utilizando taxis hacia diferentes destinos. El problema de optimización relacionado consiste en determinar el número apropiado de taxis y distribuir los pasajeros de forma eficiente, para minimizar el costo global de los recorridos.

La distribución de pasajeros está restringida a la cantidad máxima de personas que pueden viajar en un mismo taxi. Esta constante puede ser fácilmente parametrizable, permitiendo aplicar el problema a diferentes realidades.

El problema fue originalmente concebido para modelar el transporte en taxis en la ciudad de Montevideo, Uruguay, aunque el algoritmo no restringe su aplicabilidad a otros escenarios, tal como se demuestra en el análisis experimental del método de planificación propuesto. Los costos asociados al transporte están modelados con una función realista que considera la tarifación basada en distancias recorridas por cada taxi. Este modelo corresponde a considerar escenarios con tráfico fluido, sin grandes embotellamientos. La incorporación de datos de tráfico en tiempo real está propuesta como una de las principales líneas de trabajo futuro.

Con respecto al problema, las siguientes consideraciones deben ser tenidas en cuenta al instanciar la formulación genérica a un determinado escenario:

- Cada taxi puede trasladar a un número limitado de pasajeros, dependiendo de las reglamentaciones propias de cada ciudad. Este parámetro debe ser ingresado como dato de entrada del algoritmo.
- El número máximo de taxis para N pasajeros es N, en el caso particular de que cada pasajero viaje en un vehículo. Esta solución representa una cota superior trivial para el costo de transporte en un determinado escenario.
- El costo de un taxi está dado por la suma del costo inicial conocido popularmente como "bajada de bandera", más el costo determinado por la distancia recorrida desde el origen al destino. No se consideran otros posibles costos tales como esperas, propinas o cargos extras por equipaje.

II-B. Formulación Matemática

La formulación matemática del problema de viajes compartidos en taxis se presenta a continuación

Dados los siguientes elementos:

- Un conjunto de pasajeros $P = \{p_1, p_2, \ldots, p_N\}$; que parten de un punto geográfico común O y desean trasladarse hacia un conjunto de destinos potencialmente diferentes $D = \{d_1, d_2, \ldots, d_N\}$;
- Un conjunto de automóviles con taxímetro (taxis) $T = \{t_1, t_2, \ldots, t_M\}$; con $M \leq N$; donde la capacidad de cada taxi está acotada por un valor C_{MAX} (número máximo de pasajeros que puede transportar) y cada taxi tiene asociada una función $C : T \rightarrow \{0, 1, \ldots, C_{MAX}\}$ que indica cuántos pasajeros hacen uso de un taxi en un determinado viaje.
- Una matriz simétrica M de dimensión $(N+1) \times (N+1)$ que especifica las distancias entre cada uno de los puntos geográficos involucrados en el problema (un punto origen y N puntos destino);
- Una función de costo del transporte c_T determinada por los parámetros B constante (costo inicial, *bajada de bandera*) y el costo por distancia $c(dist(O, d_i))$.

El problema consiste en hallar una planificación, es decir una función $f : P \rightarrow T$ para transportar los N pasajeros en K taxis ($K \leq M$) que determine la asignación de pasajeros a taxis y el orden en que serán trasladados a los respectivos destinos, minimizando la función de costo total (CT) de la asignación, dada en la Ecuación 1.

$$CT = \sum_{t_i} (B + \sum_{j=1}^{C(t_i)} c(dist(d_{j-1}, d_j)) \qquad (1)$$

En la formulación presentada previamente se busca optimizar costo global del viaje completo. Diversas estrategias pueden aplicarse para calcular el costo por pasajero [6], por ejemplo: i) pagar por distancia y repartir equitativamente el costo de bajada de bandera, y cada pasajero j participante en un viaje compartido en el taxi t_i deberá pagar un valor de $B/C(t_i) + dist(d_{j-1}, d_j)$ por el transporte; ii) pagar una tarifa plana independientemente de las distancias, y cada pasajero en el taxi t_i deberá pagar $(B + \sum_{j=1}^{C(t_i)} c(dist(d_{j-1}, d_j))/C(t_i))$, etc.

El problema de planificación de viajes compartidos es NP-difícil, al ser una variante del problema de *car pooling* [4]. Por este motivo, para resolver instancias con dimensiones realistas en tiempos reducidos es necesario utilizar heurísticas o metaheurísticas [5].

III. Trabajos relacionados

Esta sección introduce algunos trabajos relacionados al problema propuesto. Tal como se referencia en la introducción, el problema del *car pooling* ha tenido gran interés público en los últimos años. Los beneficios que presenta han motivado diversas investigaciones en el área que aplican diversas metodologías y heurísticas.

III-A. Taxi Pooling

El problema de *taxi pooling*, una variante particular del *car pooling*, ha sido estudiado desde diferentes perspectivas en la literatura del área. Desde el punto de vista de la compañía de taxis, Xin et al. [7] estudiaron el problema de los k-taxis, que propone optimizar el costo total de k taxis que atienden pedidos de clientes en línea. La estrategia propuesta consiste en indicar a los dos taxis más cercanos a un determinado pedido

que se dirijan hacia el mismo para intentar atenderlo. Debido a esta competencia entre taxis, se evita el problema donde un taxi siempre está ocupado mientras otros se encuentran inactivos. Los resultados muestran que el problema de los k-taxis, como generalización del problema de k-servidores [8], alcanza un cociente de k entre su desempeño y el de un algoritmo óptimo que conoce toda la secuencia de pedidos de antemano.

Tratando de balancear los intereses de las compañias y los usuarios, Ma et al. [9] diseñaron un sistema dinámico para compartir taxis, combinando un algoritmo de búsqueda para encontrar taxis candidatos a satisfacer un pedido, y un algoritmo de planificación que inserta el viaje en el itinerario del taxi que incurra en la menor distancia adicional al atender ese pedido. El algoritmo de planificación chequea cada inserción posible para un nuevo viaje, verifica que satisfaga las restricciones temporales impuestas, y elige para insertar aquél con menor incremento de distancia asociada. Para mejorar el desempeño computacional del algoritmo se describe una estrategia perezosa que utiliza resultados previamente calculados, de forma de postergar lo más posible los cálculos de caminos más cortos hasta que sean necesarios. Usando una base de datos de trayectorias de GPS generadas por 33.000 taxis durante 3 meses en la ciudad de Beijing, el sistema descripto alcanzó un ahorro en la distancia de un 13 % respecto a los viajes individuales, además de atender un 25 % más de pedidos, en un escenario simulado donde la proporción de pedidos contra taxis era de 6 a 1.

Más relacionado con el problema que se presenta en este artículo, que se enfoca principalmente en los intereses del cliente, Tao et al. [10] propusieron dos heurísticas ávidas para optimizar los costos en viajes compartidos de taxis. Un algoritmo se aplica para el caso de un origen a muchos destinos y el otro para el escenario opuesto. Los resultados de una prueba piloto realizada en Taipei con 10 taxis y 798 pasajeros muestran en promedio un porcentaje de éxito en los emparejamientos de un 60.3 %. Sin embargo, los resultados reportados sobre la mejora en el consumo de combustible son en términos absolutos, no en porcentaje, por lo que es difícil extraer información útil para comparar. El problema de un origen a muchos destinos es el que se presenta en este artículo, por lo que es de particular interes comparar el desempeño del algoritmo evolutivo propuesto frente a la técnica ávida.

El análisis de trabajo relacionado muestra que el problema de viajes compartidos en taxis es de interés en la actualidad, debido a que impacta directamente tanto en el medioambiente como en la economía. Sin embargo, las soluciones centradas en el usuario son escasas en la literatura, por lo que hay lugar para el desarrollo y mejora de aplicaciones enfocadas en maximizar el ahorro de los clientes y reducir el tráfico de vehículos a través de los viajes compartidos en taxis.

III-B. Planificación en línea de vehículos compartidos

Mas allá de las investigaciones académicas en el área, existen en la actualidad diversas aplicaciones interactivas que presentan de forma amigable una solución al problema del *car pooling*. Entre las mas conocidas se encuentra la disponible en el sitio web Carpooling (*http://carpooling.com*). Esta solución permite encontrar compañeros de viajes para compartir gastos y ahorrar dinero. El servicio de Carpooling se orienta a personas que desean compartir autos propios, y no taxis, como en nuestra investigación. De todas formas, existen analogías importantes con nuestra propuesta, como la limitación en la cantidad de pasajeros por vehículo.

En cuanto al *taxi pooling*, existen también soluciones similares para usuarios finales en línea. Una de ellas es la disponible en el sitio web Carpling (*http://carpling.com*). La idea detrás de este servicio es similar a la de Carpooling, pero con un enfoque mayoritario en los taxis. A grandes rasgos, busca encontrar personas que desean viajar a lugares cercanos en taxi, y compartir gastos. Esta solución en línea se diferencia de la que se presenta en este artículo, ya que se limita a insertar viajes que estén completamente incluídos en otros, sin explorar otras posibles soluciones que de todas formas minimizen el costo total. Adicionalmente, permite evaluar a los usuarios, formando un ránking de los usuarios más activos y confiables.

En cuanto al problema presentado en este informe, no existen en la actualidad servicios en línea similares. Existen aplicaciones web con ciertos puntos en común, aunque no con las particularidades de la presente investigación.

IV. Algoritmos evolutivos

Los Algoritmos Evolutivos (AE) son técnicas estocásticas que emulan el proceso de evolución natural de las especies para resolver problemas de optimización, búsqueda y aprendizaje [11]. En los últimos 30 años, los AE han sido exitosamente aplicados para resolver problemas de optimización subyacentes a problemas complejos del mundo real en múltiples áreas de aplicación [5].

Un AE es una técnica iterativa (cada iteración se denomina *generación*) basada en emular el proceso de evolución natural de los seres vivos. En cada generación se aplican operadores estocásticos sobre un conjunto de individuos (la *población*). Inicialmente, la población se genera aplicando un procedimiento aleatorio o utilizando una heurística específica para el problema a resolver. Cada individuo en la población codifica una solución tentativa al problema estudiado, y tiene un valor de *fitness*, que es una medida relacionada con la función objetivo del problema de optimización, dado por una función de evaluación que determina su adecuación para resolver el problema. El objetivo del AE es mejorar el fitness de los individuos en la población. Este objetivo se logra mediante la aplicación iterativa de *operadores evolutivos*, como la *recombinación* de partes de dos individuos y la *mutación* aleatoria de su codificación, que son aplicados a individuos seleccionados según su fitness, guiando al AE a soluciones tentativas de mayor calidad. El Algoritmo 1 presenta el esquema genérico de un AE que trabaja sobre una población *P*.

El criterio de parada de la iteración usualmente involucra un número determinado de generaciones, una cota de calidad sobre el mejor valor de fitness hallado, o la detección de una situación de convergencia. Políticas específicas se utilizan para seleccionar el grupo de individuos para participar en la recombinación (la *selección*) y para deteminar cuáles nuevos individuos serán insertados en la población en cada nueva generación (el *reemplazo*). Finalmente, el AE retorna la mejor solución hallada en el proceso iterativo, tomando en cuenta la función de fitness considerada para el problema.

Algorithm 1 Esquema de un Algoritmo Evolutivo.

```
1: inicializar(P(0))
2: t ← 0 {contador de generación}
3: while not criterio de parada do
4:    evaluar(P(t))
5:    padres ← selección(P(t))
6:    hijos ← operadores de variación(padres)
7:    newpop ← reemplazo(hijos, P(t))
8:    t++
9:    P(t) ← newpop
10: end while
11: retornar mejor individuo hallado
```

Los detalles de diseño e implementación del AE desarrollado en este trabajo para resolver el problema de planificacion de viajes compartidos de taxis se presentan en la Sección V.

V. Diseño e implementación del AE propuesto

Esta sección presenta los detalles de diseño e implementación del AE

V-A. La bibioteca Mallba

El AE propuesto para el problema de planifiación de viajes compartidos en taxis fue desarrollado utilizando Malva [12], una implementación actualizada de la biblioteca Mallba [13].

Mallba es una biblioteca para optimización combinatoria desarrollada en C++ que incluye métodos exactos, técnicas heurísticas y metaheurísticas. Todos los algoritmos en Mallba son implementados en base a *esqueletos*, similares a un patrón genérico de nombre *strategy*. Un *esqueleto* es un algoritmo que implementa un algoritmo de optimización (por ejemplo, AE) en forma de *template*, que debe ser completado para aplicarse a un problema concreto.

El diseño de un esqueleto se basa en la separación de dos conceptos: las *características del problema* (incluyendo la función a optimizar y detalles sobre cómo manipular las soluciones tentativas) que son proporcionadas por el usuario, y las *técnicas de resolución*, que son provistas de modo abstracto por la biblioteca. Los esqueletos se implementan como un conjunto de clases requeridas y clases provistas:

- Las *clases provistas* implementan aspectos internos de los algoritmos, de modo independiente a los problemas a resolver, en los archivos con extensión *pro.cc*. Las clases provistas más importantes son *Solver* (el algoritmo) y *SetUpParams* (para fijar los parámetros del método de resolución).
- Las *clases requeridas* especifican información relacionada al problema, implementadas en los archivos con extensión *req.cc*. Cada esqueleto incluye las clases requeridas *Problem* y *Solution* que encapsulan las entidades necesarias para resolver el problema. Otras clases pueden ser requeridas dependiendo del método de resolución empleado.

El AE presentado en este trabajo fue desarrollado extendiendo el esqueleto newGA de Mallba.

V-B. Representación de soluciones

Las soluciones del problema son representadas como tuplas de largo $2N-1$ donde N es el número de pasajeros de la instancia del problema a resolver. Las tuplas contienen los enteros del 1 al N, que representan a cada uno de los pasajeros, y $N-1$ separadores (ceros), para distinguir grupos de pasajeros asignados a diferentes taxis. Cada grupo de dígitos distintos de cero representan por lo tanto a un taxi, que transportará a su destino a cada uno de los pasajeros, respetando el orden en que aparecen en la secuencia. La Figura 1 muestra un ejemplo de representación de solución para una instancia con $N = 5$.

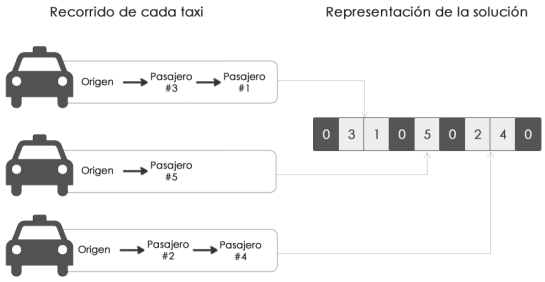


Figura 1. Ejemplo de representación de una solución para el problema de planificación de viajes compartidos en taxis.

Dada la representación elegida y la realidad del problema, surgen las siguientes restricciones para cada tupla:

- El número de dígitos consecutivos mayores que cero está limitado por la máxima cantidad de pasajeros permitidos por taxi.
- Cada número mayor que cero debe aparecer una y sólo una vez, para asegurar que cada pasajero está asignado a un único taxi.
- Finalmente, se observa que dos o más ceros consecutivos cumplen el mismo rol que un único cero.

V-C. Generación de la población inicial

La población inicial del AE es generada mediante un algoritmo constructivo descripto en el Algoritmo 2.

Algorithm 2 Inicialización aleatoria de las soluciones del AE

```
for j = 1 to 2N−1 do
   sol[j] = 0; {inicializar solución con ceros}
end for
for i = 1 to N do
   repeat
      posicion = random(2N−1);
   until (sol[posicion]==0) {sortear una posición libre}
   sol[posicion] = i;
end for
return sol;
```

La generación aleatoria aplicando el Algoritmo 2 puede violar alguna de las restricciones impuestas por el mecanismo de representación. Para resolver este inconveniente se aplica un procedimiento de corrección a las soluciones generadas (Algoritmo 3), desplazando ceros consecutivos de forma de romper, en un punto aleatorio, las secuencias de dígitos mayores que cero que no cumplen con el largo máximo permitido.

Algorithm 3 Procedimiento de corrección de soluciones

```
seguir = true;
while seguir do
  contador = 0;
  i = 0;
  while contador<= C_MAX and i < N do
    if sol[i] ≠ 0 then
      contador++;
    else
      contador = 0;
    end if
    i++;
  end while
  if cantidad> C_MAX then
    m = random((i-C_MAX)+1,i-1);
    p = encontrarPrimerCeroConsecutivo();
    intercambiar(p,m);
  else
    seguir=false;
  end if
end while
```

V-D. *Función de fitness*

El objetivo del problema es minimizar el gasto total de los pasajeros, las soluciones con menor costo total serán las más adecuadas para resolver el problema. Para transformar el problema de minimización de costo en un problema de maximización de fitness, se utiliza el inverso de la función de costo total de una solución, tal como fue presentada en la Ecuación 1.

V-E. *Operadores*

Debido a las restricciones en la representación explicadas en la Sección V-B, los operadores usados por el AE deben ser adaptados para aplicarse al problema.

1) Selección: Se utilizó el operador de *selección proporcional*, que asigna a cada individuo una probabilidad de selección dada por el cociente entre su valor de fitness y la suma del fitness de los individuos en la población (Ecuación 2)

$$P_i = \frac{fitness(i)}{\sum_{j=1}^{N} fitness(j)} \quad (2)$$

2) Recombinación: Se utilizó una versión modificada del operador de *Cruzamiento Basado en la Posición (Position Based Crossover - PBX)* (Algoritmo 4).

Algorithm 4 Cruzamiento Basado en la Posición - *PBX*

1: Seleccionar aleatoriamente varias posiciones del padre 1.
2: generar parcialmente el hijo 1, copiando los valores (*alelos*) de los las posiciones elegidas del padre 1.
3: Marcar los alelos del padre 2 que ya fueron seleccionados en el padre 1.
4: Desde el inicio del padre 2, seleccionar secuencialmente el siguiente alelo que no haya sido marcado, y ponerlo en la primer posición libre del hijo 1, desde el comienzo.

Para evitar violar las restricciones de codificación, se aplica el procedimiento de corrección al completar el cruzamiento PBX, de forma de asegurar que se cumplan las restricciones impuestas. La Figura 2 muestra un ejemplo del PBX para soluciones con 5 pasajeros y un máximo de 4 por taxi.

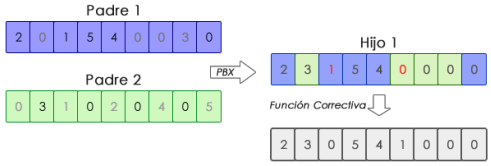


Figura 2. Ejemplo de cruzamiento PBX aplicado al problema.

3) Mutación: Se utilizó el operador de *Mutación por Intercambio (Exchange Mutation - EM)*. Es un operador muy simple de implementar, que sortea aleatoriamente dos posiciones en la solución y las intercambia. De ser necesario, el procedimiento de corrección se aplica posteriormente a la EM, para corregir posibles conflictos en las soluciones mutadas.

VI. Resultados experimentales

Esta sección reporta el análisis experimental del algoritmo evolutivo propuesto sobre un conjunto de instancias realistas del problema.

VI-A. *Plataforma de desarrollo y ejecución*

El AE se desarrolló en C++, utilizando la biblioteca Mallba y el compilador g++ versión 4.7.2 (Ubuntu/Linaro 4.7.2-2ubuntu1). El análisis experimental se llevó a cabo en un equipo con procesador Intel Core i5, CPU M 430 a 2.27GHz, 4GB de memoria RAM, y sistema operativo Ubuntu Linux 12.04 de 64 bits.

VI-B. *Instancias del problema*

Para evaluar el AE propuesto, se generó un conjunto de instancias realistas del problema de planifiación de viajes compartidos en taxis. Se utilizó un enfoque metodológico específico para la generación de instancias realistas del problema, siguiendo los lineamientos presentados en los trabajos relacionados y utilizando servicios disponibles para recabar información sobre pedidos de taxi, mapas, y tarifas, incluyendo:

- *Generador de Pedidos de Taxis* (*Taxi Query Generator-TQG*), una herramienta que utiliza información de una base de datos de trayectorias de taxis, obtenidas a través de los dispositivos GPS instalados durante una semana en 10.357 taxis en la ciudad de Beijing, para generar pedidos de taxis realistas. Los datos son una muestra de los utilizados por Ma *et al.* [9] tal como se mencionó en la Sección III. TQG genera una lista con las coordenadas de origen y destino de viajes individuales. Para obtener instancias aplicables al problema presentado en este artículo, se implementó un script que agrupa viajes que comienzan en orígenes cercanos obtenidos a través del TQG, y devuelve instancias del problema de un origen a muchos destinos.

- *QGIS Desktop*, un sistema de información geográfica gratuito y de código abierto, utilizado para crear, editar, visualizar y publicar material geoespacial [14]. QGIS se utilizó para transformar la lista de coordenadas obtenidas a la salida del script de agrupación en un archivo KML (Keyhole Markup Language), que permite ser visualizado utilizando Google Maps/Google Earth [15], de forma de tener una representación visual del origen y destinos de la instancia en un mapa.
- La interfaz para aplicaciones de *TaxiFareFinder* [16], que fue utilizada para obtener la matriz de costos de cada instancia del problema generada, junto a los precios correspondientes a la bajada de bandera. Cada par de coordenadas de una determinada instancia del problema es enviado a la interfaz de TaxiFareFinder para obtener el costo entre cada punto.

Siguiendo el enfoque propuesto, se diseñó un benchmark compuesto por seis instancias del problema, utilizando información realista.

VI-C. Ajuste paramétrico

Debido a la cualidad estocástica de los AEs, es necesario ajustar sus parámetros previamente al análisis experimental. Para los experimentos de configuración pramétrica se utilizaron tres instancias del problema con diferentes características, fijando el máximo número de pasajeros permitidos en un mismo taxi en 4. El ajuste se focalizó en estudiar tres parámetros del AE: tamaño de la población (#*P*), probabilidad de recombinación (p_C) y probabilidad de mutación (p_M). Para cada parámetro fueron estudiados los siguientes valores candidatos: #*P*: 150, 200, 250; p_C: 0.6, 0.75, 0.95; y p_M: 0.001, 0.01, 0.1.

Se estudiaron todas las combinaciones de valores sobre las tres instancias de calibración, realizando 20 ejecuciones independientes del AE en cada caso, con 2000 generaciones en cada ejecución, para resolver cada problema.

Los resultados se muestran gráficamente en la Figura 3, reportando el costo promedio obtenido en las tres instancias para cada una de las combinaciones de parámetros. Las tablas de resultados completos se reportan en http://www.fing.edu.uy/inco/grupos/cecal/hpc/AG-Taxi.

Recordando que se trata de un problema de minimización, aquellas configuraciones de parámetros que alcancen valores menores de costo serán preferidas. Para los 3 tests se observa que los mejores resultados se obtienen con el máximo tamaño de población (250), la mínima probabilidad de recombinación (0.6), y la máxima probabilidad de mutación (0.1).

VI-D. Evaluación experimental

El análisis experimental se enfocó en evaluar el desempeño del AE y la calidad de las soluciones alcanzadas. Se utilizaron seis instancias del problema (diferentes a las utilizadas en los experimentos de calibración, para evitar sesgos en los resultados), que corresponden a escenarios de entre 25 y 50 pasajeros. Estas dimensiones son razonablemente mayores a lo que se puede esperar en instancias generadas por usuarios finales, evaluando el AE sobre instancias complejas.

1) Resultados numéricos: La Tabla I reporta los resultados del AE para las seis instancias del problema estudiadas. Las ejecuciones fueron realizadas con la configuración paramétrica determinada en los experimentos de calibración.

Debido a que la aplicación tiene un fuerte enfoque hacia el usuario final, es importante mantener un balance entre la calidad de las soluciones y el tiempo de respuesta. De acuerdo al trabajo de Cantú-Paz y Goldberg [17], generalmente es preferible una única ejecución con un número mayor de generaciones, frente a múltiples ejecuciones más pequeñas. En consecuencia, el AE fue evaluado usando diferentes criterios de parada en términos del número de generaciones (10000, 25000, 50000 y 75000).

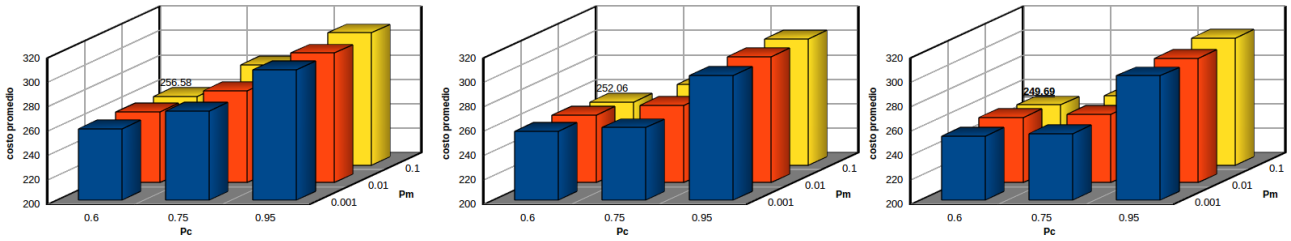
Se realizaron 20 ejecuciones independientes del AE con cada uno de los distintos criterios de parada, para cada una de las instancias del problema. Para analizar las soluciones obtenidas se aplicó el test de Shapiro-Wilk (S-W) [18] para establecer si los valores de costo obtenidos seguían o no una distribución normal y poder comparar apropiadamente las medias/medianas. Shapiro-Wilk ha demostrado un mejor desempeño para el chequeo de normalidad entre los tests estadísticos de normalidad más utilizados [19].

La Tabla I reporta los siguientes valores: mejor valor de costo alcanzado ($min(\overline{CT})$); promedio de los mejores valores de costo alcanzados ($\overline{CT}$); varianza de los mejores valores de costo alcanzados ($\sigma^2(CT)$); p-valor del test de Shapiro-Wilk sobre los valores de costo (*p-valor S-W*); generación en que se alcanzó la mejor solución ($min(gen)$); generación promedio en que se alcanzó la mejor solución ($\overline{gen}$); tiempo de ejecución mínimo ($min(T)$); tiempo de ejecución promedio ($\overline{T}$); y varianza del tiempo de ejecución ($\sigma^2(T)$). Todos los tiempos reportados se miden en segundos.

Los resultados númericos reportados en la Tabla I demuestran que para un nivel de significación de $\alpha = 0.05$ en el test de Shapiro-Wilk para normalidad sobre los valores de costo, todas las ejecuciones cumplen $p - valor > \alpha$, de donde es posible concluir que no puede descartarse la hipótesis nula del test de Shapiro-Wilk, que propone que los valores de costo obtenidos siguen una distribución normal.

Observando los promedios de los mejores valores de costo alcanzados, es claro que ejecuciones con un mayor número de generaciones llevan a mejores soluciones. Sin embargo, la mejora alcanzada depende de las características propias de la instancia que se resuelve. Por ejemplo, para la instancia #3 se observa un 22.5 % de mejora en promedio entre las soluciones obtenidas con 75.000 generaciones y las obtenidas con 10.000 generaciones, mientras que para la instancia #6 esta mejora es de un 10.6 % en promedio.

2) Comparación contra un algoritmo ávido que resuelve el problema: Un algoritmo ávido (o *greedy*) es un método de construcción de soluciones que toma la decisión localmente óptima en cada paso, con la esperanza de llegar a un óptimo global del problema. Como se mencionó en III, es relevante comparar el desempeño del AE frente a un algoritmo ávido, en términos de la calidad de las soluciones alcanzadas y de la eficiencia computacional. Con este propósito, se implementó un algoritmo ávido sencillo para resolver el problema de planificación de viajes compartidos abordado en este artículo.



(a) Tamaño de la población = 150 (b) Tamaño de la población = 200 (c) Tamaño de la población = 250

Figura 3. Costo promedio obtenido en las instancias de calibración con diferentes configuraciones de parámetros (los valores más bajos se prefieren).

Tabla I. EVALUACIÓN EXPERIMENTAL: DESEMPEÑO DEL AE

test	#gen.	$min(CT)$	$\overline{CT}$	$\sigma^2(CT)$	*p-valor S-W*	$min(gen)$	$\overline{gen}$	$min(T)$	$\overline{T}$	$\sigma^2(T)$
instancia #1	10000	421.0	471.6	1343.6	0.4	982.0	7877.1	13.5	13.6	7.1e-03
	25000	378.6	411.5	283.8	0.3	12230.0	2.1e+04	33.6	34.3	0.2
(32 pasajeros)	50000	372.4	397.3	187.4	0.6	23271.0	4.3e+04	68.1	70.0	2.1
	75000	366.8	388.1	112.5	0.1	20836.0	5.7e+04	101.3	102.7	1.3
instancia #2	10000	118.1	134.9	139.6	0.1	2214.0	7605.6	10.3	10.5	1.1e-02
	25000	113.1	128.7	58.8	0.5	10448.0	1.8e+04	25.8	26.5	0.2
(26 pasajeros)	50000	113.3	124.2	40.4	0.7	11971.0	3.2e+04	52.9	56.6	27.7
	75000	115.2	123.2	29.2	0.7	11825.0	5.0e+04	76.8	78.9	0.8
instancia #3	10000	367.5	464.4	2994.0	0.1	2750.0	7070.5	15.2	15.4	1.1e-02
	25000	372.6	427.8	1703.1	0.2	1292.0	1.6e+04	38.1	38.4	5.2e-02
(35 pasajeros)	50000	336.0	367.2	322.9	0.6	19436.0	4.2e+04	76.6	77.5	0.5
	75000	335.7	360.0	183.6	0.6	41013.0	6.5e+04	115.1	119.4	9.8
instancia #4	10000	233.9	258.4	237.2	0.4	4463.0	7882.6	18.7	19.1	5.5e-02
	25000	209.3	240.9	273.9	0.7	16049.0	2.1e+04	46.1	47.2	0.2
(41 pasajeros)	50000	211.4	228.6	94.9	0.7	22717.0	4.2e+04	92.1	96.9	22.0
	75000	199.4	222.9	133.8	1.0	33033.0	6.4e+04	140.0	142.0	0.8
instancia #5	10000	521.9	613.9	3928.9	0.6	226.0	7029.2	18.7	19.0	5.1e-02
	25000	468.6	530.0	1764.0	0.1	4259.0	2.0e+04	46.8	51.3	32.6
(41 pasajeros)	50000	464.5	491.9	417.9	0.2	16940.0	4.0e+04	93.7	95.1	0.7
	75000	454.9	483.0	252.6	0.2	39926.0	6.0e+04	141.7	143.2	4.7
instancia #6	10000	401.6	445.8	586.0	0.8	3728.0	8679.0	15.2	15.6	7.7e-02
	25000	397.1	417.4	198.6	0.2	11685.0	20066.0	38.1	38.7	0.4
(35 pasajeros)	50000	368.7	403.8	246.2	0.7	15776.0	4.0e+04	76.9	78.1	1.9
	75000	380.0	398.4	98.0	0.4	42089.0	6.3e+04	115.7	119.8	5.7

Tabla II. EVALUACIÓN EXPERIMENTAL: COMPARACIÓN ENTRE AE Y ALGORITMO ÁVIDO

test	*algoritmo*	$min(CT)$	$\overline{CT}$	$min(T)$	$\overline{T}$	*p-valor M-W*	mejora sobre alg. ávido (max)	mejora sobre alg. ávido (prom.)
	AE_{10000}	421.0	471.6	13.4	13.6	1.5e-05	19.6 %	10.0 %
instancia #1	AE_{25000}	378.6	411.5	33.6	34.3	0.0	27.8 %	21.5 %
(32 pasajeros)	AE_{50000}	372.4	397.3	68.1	70.0	0.0	28.9 %	24.2 %
	AE_{75000}	366.8	388.1	101.3	102.7	0.0	30.0 %	25.9 %
	alg. ávido	524.0	524.0	0.0	0.0	-	-	-
	AE_{10000}	118.1	134.9	10.3	10.5	0.0	39.1 %	30.4 %
instancia #2	AE_{25000}	113.1	128.7	25.8	26.5	0.0	41.7 %	33.6 %
(26 pasajeros)	AE_{50000}	113.3	124.2	52.9	56.6	0.0	41.6 %	36.0 %
	AE_{75000}	115.2	123.2	76.8	78.9	0.0	40.6 %	36.4 %
	alg. ávido	193.9	193.9	0.0	0.0	-	-	-
	AE_{10000}	367.5	464.4	15.2	15.4	1.5e-05	29.6 %	11.0 %
instancia #3	AE_{25000}	372.6	427.8	38.1	38.4	0.0	28.6 %	18.0 %
(35 pasajeros)	AE_{50000}	336.0	367.2	76.6	77.5	0.0	35.6 %	29.6 %
	AE_{75000}	335.7	360.0	115.1	119.4	0.0	35.7 %	31.0 %
	alg. ávido	522.0	522.0	0.0	0.0	-	-	-
	AE_{10000}	233.9	258.4	18.7	19.1	1.5e-05	14.1 %	5.1 %
instancia #4	AE_{25000}	209.3	240.9	46.1	47.2	0.0	23.1 %	11.5 %
(41 pasajeros)	AE_{50000}	211.4	228.6	92.1	96.9	0.0	22.4 %	16.0 %
	AE_{75000}	199.4	222.9	140.0	142.0	0.0	26.7 %	18.1 %
	alg. ávido	272.2	272.2	0.0	0.0	-	-	-
	AE_{10000}	521.9	613.9	18.7	19.0	3.0e-02	9.6 %	-6.3 %
instancia #5	AE_{25000}	468.6	530.0	46.8	51.3	1.5e-04	18.8 %	8.2 %
(41 pasajeros)	AE_{50000}	464.5	491.9	93.7	95.1	0.0	19.5 %	14.8 %
	AE_{75000}	454.9	483.0	141.7	143.2	0.0	21.2 %	16.4 %
	alg. ávido	577.3	577.3	0.0	0.0	-	-	-
	AE_{10000}	401.6	445.8	15.2	15.6	3.0e-02	12.7 %	3.1 %
instancia #6	AE_{25000}	397.1	417.4	38.1	38.7	0.0	13.7 %	9.3 %
(35 pasajeros)	AE_{50000}	368.7	403.8	76.9	78.1	0.0	19.9 %	12.3 %
	AE_{75000}	380.0	398.4	115.7	119.8	0.0	17.4 %	13.4 %
	alg. ávido	460.2	460.2	0.0	0.0	-	-	-

El algoritmo ávido implementado se basa en aplicar una técnica de agrupamiento simple e intuitiva. Toma un taxi, y agrega el destino más cercano al taxi actual hasta completarlo (en cuyo caso forma un nuevo taxi), o hasta que todos los pasajeros hayan sido asignados. Una excepción ocurre cuando el costo de añadir un nuevo destino al taxi actual es mayor al costo de asignar un nuevo taxi que transporte únicamente a ese pasajero. En este caso, se forma un nuevo taxi y se "cierra" el anterior. Este comportamiento emula una estrategia intuitiva para resolver el problema con un enfoque de agregación de estrategias de decisión localmente óptimas, y se aproxima a lo que un grupo de usuarios humanos pueden idear de manera simple para resolver el problema. La estructura básica del algorimo ávido implementado para la comparación de los resultados obtenidos por el AE se muestra en el Algoritmo 5.

Algorithm 5 Algoritmo ávido para compartir taxis

```
M = C_MAX;
taxi = 1;
contador = 0;
while pasajerosSinAsignar() do
  if (contador == 0) then
    p1 = destinoMasCercanoDesdeOrigen();
    agregarPasajero(p1,taxi,solucion);
    contador++;
  else if (contador ≥ M) then
    contador = 0; {El taxi actual está completo}
    taxi++;
  else
    p2 = vecinoMasCercano(p1);
    if (costo(p1,p2) ≤ costo(origen,p2) + B) then
      agregarPasajero(p2,taxi,solucion);
      p1 = p2;
      contador++;
    else
      taxi++; {Comenzar con un nuevo taxi}
      contador = 0;
    end if
  end if
end while
return solucion;
```

La Tabla II reporta el análisis comparativo entre los resultados calculados por el AE y el algoritmo ávido, para las instancias de la evaluación experimental. Para analizar la significancia estadística de la comparación, se aplicó el test bilateral U de Mann-Whitney [20] para establecer si los resultados obtenidos con el AE tienen una diferencia significativa con los calculados por el algoritmo ávido. La columna *p-valor M-W* en la Tabla II reporta el p-valor del test de Mann-Whitney sobre los valores de costo calculados por el AE. Las restantes columnas corresponden a las explicadas para la tabla previa. Todos los tiempos reportados se miden en segundos.

El test U de Mann-Whitney aplicado sobre las distribuciones de resultados muestra que existe una diferencia significativa entre las soluciones obtenidas por el AE y aquellas alcanzadas por el algoritmo ávido. Para todos los casos, excepto para la instancia #5, una única ejecución de 25.000 generaciones fue suficiente para obtener una mejora sobre la solución obtenida por el algoritmo ávido. Para la instancia #2, ejecutando por solo 25.000 generaciones, se alcanzó un **33.6 %** de mejora en promedio y **41.7 %** en el mejor caso. Para la instancia #3 estos porcentajes son de **18.0 %** en promedio y **28.6 %** en el mejor caso. Sin embargo, permitiendo al AE evolucionar por 75.000 generaciones, la mejora sube a **31.0 %** en promedio y **35.7 %** en el mejor caso.

La Figura 4 resume los promedios de los mejores valores y valores promedios de mejora (diferencia porcentual) con respecto al algoritmo ávido, entcontrados por el AE utilizando 75.000 generaciones.

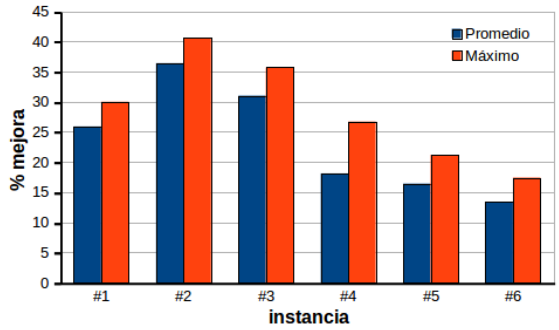


Figura 4. Porcentajes de mejora con respecto al algoritmo ávido.

Dado que el algoritmo ávido ejecuta en un tiempo negligible, es importante evaluar el tiempo requerido por el AE para calcular mejores resultados. Este tiempo de respuesta se verá reflejado cuando el usuario final utilice el AE en una aplicación web o móvil. Para este fin, se diseñaron experimentos de 20 ejecuciones independientes, sobre las mismas 6 instancias anteriores, programando el AE con el siguiente criterio de parada: si supera al algoritmo ávido en un 20 %, si no mejora la solución por 20.000 generaciones (se asume que la búsqueda se estancó), o al cabo de 100.000 generaciones.

Las Figuras 5(a)–5(f) reportan el tiempo requerido por el AE para alcanzar soluciones de igual calidad que el algoritmo ávido, y para superarlo en un 10 % y en un 20 %, para las seis instancias del problema abordadas en en análisis comparativo. Sobre cada barra se indica el porcentaje de mejora que alcanzó el AE sobre el algoritmo ávido.

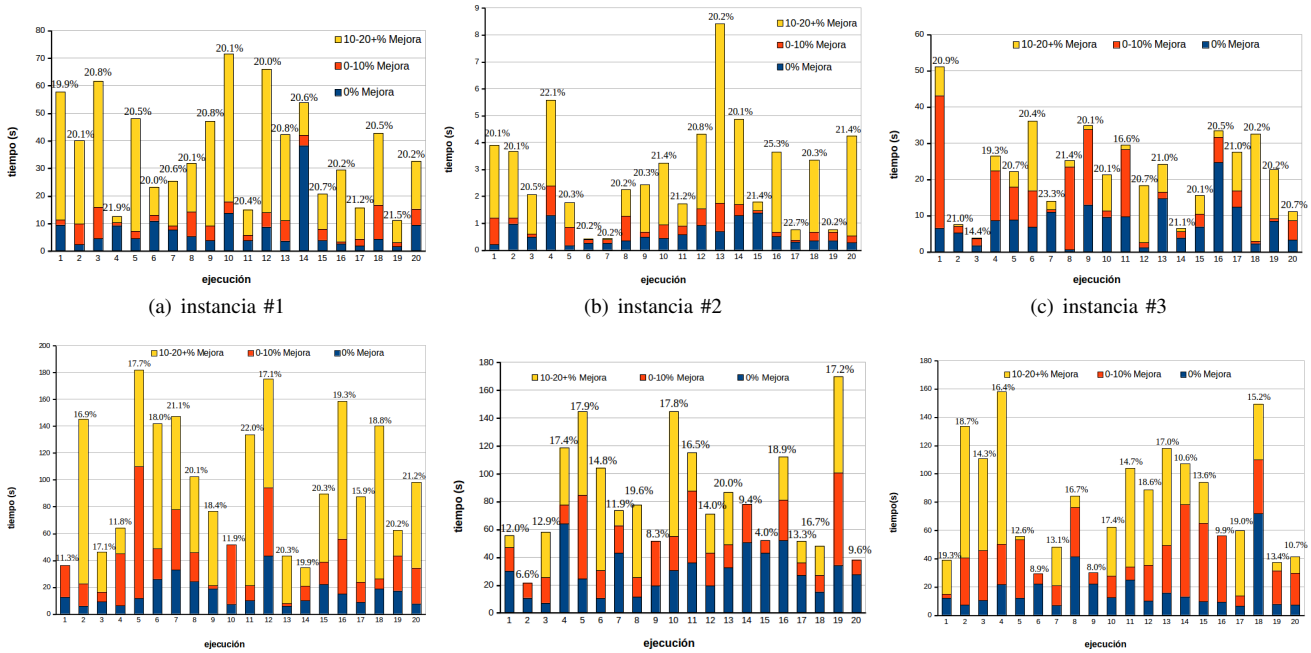
Los resultados muestran que el AE es capaz de obtener resultados similares al ávido al ejecutar por un tiempo en el entorno de los 10 segundos, y mejorarlo en más del 10 % en la mayoria de los casos, al ejecutar por un tiempo mayor. Este tiempo depende fuertemente de la instancia que se resuelve y varía desde menos de 30 segundos hasta un par de minutos.

VII. Aplicaciones Web y Móvil

Para la resolución de instancias reales del problema se desarrolló un sistema formado por una aplicación web y una aplicación móvil, al cual llamaremos de ahora en más *Planificador de Viajes Compartidos en línea*.

VII-A. Arquitectura

Una de las principales decisiones de diseño fue seleccionar una arquitectura acorde a los requerimientos del sistema. Los mismos contemplaban los siguientes puntos:



(a) instancia #1 (b) instancia #2 (c) instancia #3

(d) instancia #4 (e) instancia #5 (f) instancia #6

Figura 5. Tiempo requerido por el AE para alcanzar/mejorar al algoritmo ávido.

- Ejecución de un programa de alto peso computacional para el cálculo de la solución mediante un AE.
- Fuertes conexiones con los servicios de Google Maps para obtener las distancias entre los puntos.
- Gran volumen de procesamiento remoto, involucrando operaciones de lectura/escritura en archivos.
- Conexiones temporales con aplicaciones móviles para el cálculo de la solución.
- Necesidad de un desarrollo ágil.

En base a los puntos mencionados, se desarrolló un sistema con una arquitectura simple pero escalable, utilizando tecnologías particulares en cada uno de los componentes. Los detalles se presentan en la Figura 6.

1) Servidor Central: Se trata del componente principal del *Planificador de Viajes Compartidos*. Su rol central lo ubica como un componente imprescindible para que el sistema se comporte correctamente y sea funcional. El servidor central es el encargado de recibir las solicitudes de los usuarios web, y retornar las páginas web que el usuario tendrá en su navegador como resultado de utilizar el sistema. Por otra parte, brinda una *Interfaz de Programación de Aplicaciones (API)* para la comunicación y posterior ejecución de instancias en aplicaciones ejecutando en teléfonos móviles. En cuanto a particularidades técnicas, el servidor central está alojado en la infraestructura gratuita de *Heroku* [21] y está implementado en el lenguaje de programación dinámico *Ruby* [22], usando el framework de desarrollo *Rails* [23].

2) Aplicación Móvil: Este componente permite utilizar el *Planificador de Viajes Compartidos* desde teléfonos móviles. Particularmente, fue desarrollado para teléfonos inteligentes con sistema operativo *iOS*.

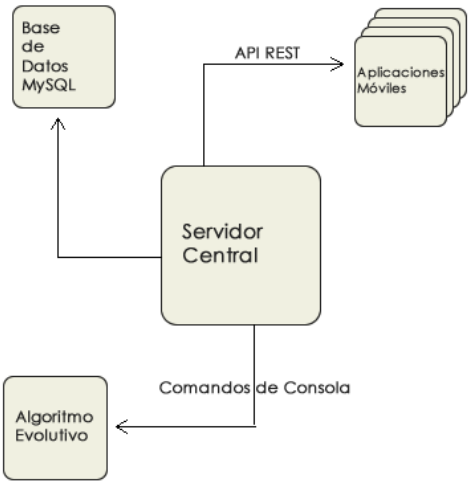


Figura 6. Arquitectura del Sistema Planificador de Viajes Compartidos.

3) Base de datos: Con el fin de optimizar las consultas realizadas al sisitema y persistirlas en el tiempo, se hace uso de una base de datos MySQL relacional. En la implementación actual la base de datos se encuentra alojada en el mismo servidor que el Servidor Central, aunque es posible migrarla a otro equipo si es necesario mejorar el desempeño computacional del *Planificador de Viajes Compartidos* como sistema.

La base de datos implementada para el *Planificador de Viajes Compartidos* cuenta con cuatro tablas principales: i) *locations*, ii) *queries*, iii) *users* y iv) *roles*, para modelar las ubicaciones ingresadas por los usuarios, las consultas realizadas, los usuarios y los roles de la aplicación web, respectivamente.

4) Algoritmo Evolutivo: La cantidad de generaciones del AE fue fijada en 15.000, de forma de balancear el desempeño del algoritmo y la usabilidad de la aplicación. Para las instancias realistas que suelen manejar los potenciales usuarios del sistema, con tamaño de entrada de unas pocas decenas de ubicaciones, el AE logra calcular una solución de alta calidad en unos pocos segundos (tal como fue demostrado en el análisis experimental presentado en la sección previa), brindando una experiencia de usuario óptima.

También se ejecuta el algoritmo ávido descripto en el Algoritmo 5, que en un tiempo despreciable permite obtener una solución de resguardo en el eventual caso en que la solución alcanzada por el AE no tenga un mejor costo. Este proceso es invisible desde el punto de vista del usuario, al cuál se le presenta la mejor solución alcanzada por alguno de los dos algoritmos.

Para la ejecución del AE encargado de brindar las soluciones para los puntos ingresados por los usuarios, se dispone de un ejecutable compilado. La comunicación entre el Servidor Central y el ejecutable se realiza mediante archivos de texto para la entrada de datos al ejecutable y para la salida. Una vez se calculan los resultados, se almacenan en la base de datos, con el objetivo de volver a tenerlos de forma rápida si el usuario desea acceder nuevamente a la consulta.

VII-B. Aplicación Web

Con el fin de acelerar el proceso de desarrollo, luego de investigar las opciones disponibles para implementar el sistema, se decidió desarrollar el Servidor Central basándose en el lenguaje de programación Ruby y el framework Rails. La aplicación web se encuentra disponible en línea en la dirección www.mepaseaste.uy. La interfaz de usuario de la aplicación desarrollada se presenta en la Figura 7.

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos. Se trata de un lenguaje interpretado en una sola pasada y su implementación oficial es distribuida bajo una licencia de software libre. Rails, por su parte, es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC). Rails trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración.

Uno de las ventajas más importantes que tiene Ruby On Rails sobre los demás entornos que estuvieron en discusión, fue la facilidad de ejecutar programas externos de consola. Complementariamente, permite publicar de forma simple una API para la comunicación externa de dispositivos móviles.

1) Modelos: En las aplicaciones web orientadas a objetos sobre bases de datos, el Modelo consiste en las clases que representan a las tablas de la base de datos. Las definiciones de las clases también detallan las relaciones entre clases con sentencias de mapeo objeto relacional. Por ejemplo, si la clase *Consulta* tiene una definición *has_many:ubicaciones*, y existe una instancia de Consulta llamada a, entonces *a.ubicaciones* devolverá un array con todos los objetos Ubicaciones cuya columna consulta_id (en la tabla comentarios) sea igual a *a.id*.

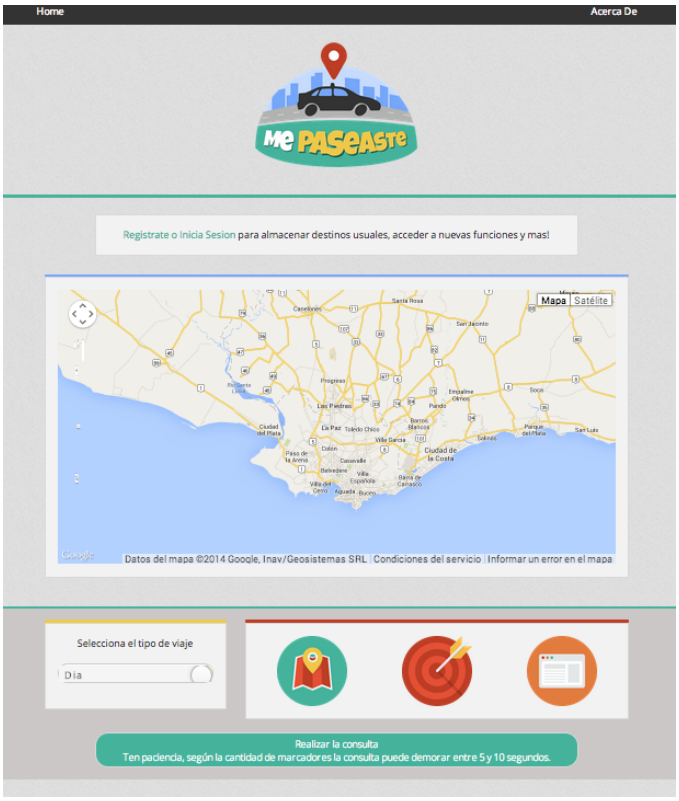


Figura 7. Interfaz de la aplicación web para el problema de compartir taxis.

Dos modelos importantes implementados en el Servidor Central son *Consulta* y *Ubicacion*. *Consulta* modela las consultas del usuario, con una serie de atributos particulares, por ejemplo si la consulta es pública o no, el número de ubicaciones, entre otros, mientras que *Ubicacion* modela las ubicaciones ingresadas, identificadas por su latitud y longitud. Además, se implementaron modelos para los usuarios y roles, que brindan funcionalidades extra a la aplicación web, tales como el inicio de sesión y el manejo de una sección de administración simple.

2) Vistas: En las aplicaciones MVC existe un componente fundamental, llamado *Vista*, que representa los datos obtenidos por el Controlador en el navegador del usuario. Usualmente, en las aplicaciones web la vista consiste en una cantidad mínima de código incluido en HTML. En el caso de la aplicación web desarrollada para la planificación de taxis, existen vistas para ingresar los puntos, y vistas para visualizar los resultados. Asimismo, también existen vistas orientadas a mejorar la experiencia de usuario, incluyendo un perfil de usuario y páginas de inicio de sesión, entre otros.

3) Controladores: En MVC, las clases del Controlador responden a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del Modelo y muestra los resultados usando las Vistas. En las aplicaciones web basadas en MVC, los métodos del controlador son invocados por el usuario usando el navegador web. En la aplicación web implementada, existen controladores para los resultados y para la gestión de los puntos ingresados.

4) Procesamiento en el navegador: Un punto no menor, es que la aplicación tiene gran cantidad de procesamiento

en el navegador del cliente. Cada vez que el usuario inserta un punto en el mapa, se ejecutan una serie de métodos asíncronos en el navegador del usuario, utilizando la tecnología Javascript/AJAX, para obtener la distancia del punto ingresado a todos los demás previamente añadidos. La secuencia de ejecución es la siguiente:

- Cuando el usuario realiza un click sobre el mapa, se almacena su latitud y su longitud en un arreglo.
- Para cada uno de los puntos ingresados anteriormente, se realiza una solicitud a la API de Google Maps con las coordenadas del punto ingresado, y las coordenadas del punto al cual se quiere calcular la distancia. El resultado se almacena en una matriz dinámica.
- Cuando el usuario presiona el botón de enviar, la matriz de distancias se envía al servidor, quien se encarga de calcular los costos y ejecutar el AE.

Por otra parte, al mostrar un resultado, también se hace uso de la API de Google Maps para graficar las rutas que debe recorrer cada taxi.

5) Costo de los Taxis: Las tarifas de taxis de las distintas ciudades son obtenidas a través de la API del servicio Taxi Fare Finder [16]. Una invocación a dicha API con las coordenadas de dos puntos devuelve la tarifa asociada a dicho viaje, con información detallada acerca del costo de bajada de bandera, de la distancia recorrida y otros costos adicionales.

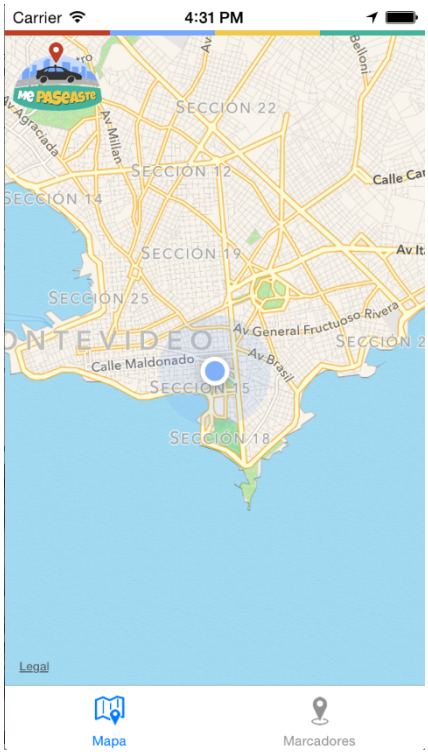


Figura 8. Interfaz de la aplicación móvil para el problema de compartir taxis.

VII-C. Aplicación Móvil

La decisión de implementar una aplicación nativa para un conjunto reducido de móviles en lugar de una versión móvil basada en un comportamiento web, pero adaptada a pantallas pequeñas, estuvo fuertemente basada en las razones de experiencia de usuario. La aplicación de planificación de viajes compartidos en taxis tiene como particularidad que despliega información en mapas, y este procesamiento consume una gran cantidad de recursos. El acceso a la aplicación web a través de dispositivos móviles genéricos es posible, pero el comportamiento de la aplicación dista de ser atractivo.

Por este motivo, se optó por desarrollar una versión de la aplicación para teléfonos inteligentes con el sistema operativo iOS de Apple. Entre los modelos soportados se encuentran los iPhones 3G, 3GS, 4, 4S, 5 y 5S. Se espera que la versión para iOS sea la primera de una serie de implementaciones para diferentes plataformas móviles.

1) Prototipo en PhoneGap: Al comenzar el desarrollo, se implementó un prototipo basado en la tecnología *PhoneGap* [24]. PhoneGap es un framework para el desarrollo de aplicaciones móviles producido por Nitobi que permite a los programadores desarrollar aplicaciones para dispositivos móviles utilizando herramientas genéricas tales como JavaScript, HTML5 y CSS3, que en nuestro caso fueron ya utilizadas para el desarrollo de la versión web. Las aplicaciones resultantes son híbridas. Por un lado, no son realmente aplicaciones nativas al dispositivo, ya que la generación de las vistas se realiza mediante un proceso análogo al realizado en los navegadores web y no con interfaces gráficas específicas de cada sistema. Por otro lado, tampoco son aplicaciones web, teniendo en cuenta que las aplicaciones son empaquetadas para poder ser desplegadas en el dispositivo incluso trabajando con el API del sistema nativo.

Al evaluar el prototipo desarrollado en PhoneGap, se pudo verificar que, ante las caraceterísticas que el sistema demandaba, una aplicación de este estilo brindaría una mala experiencia de usuario. Como ejemplos de la baja aplicabilidad práctica del prototipo se pudo verificar que la aplicación tardaba varios segundos en responder al ingreso de un único punto, y no respondía correctamente a gestos sobre el mapa.

Los pobres resultados de usabilidad y desempeño del prototipo desarrollado orientaron al desarrollo nativo como la mejor alternativa para la aplicación. Esta decisión acota el número de dispositivos para los cuales es posible utilizar el servicio, pero brinda una aplicación con una mejor experiencia de uso y un desempeño muy superior.

2) Desarrollo Nativo: La aplicación nativa desarrollada, a diferencia del prototipo, fue implementada y evaluada en un conjunto específico de dispositivos. Este análisis permitió mejorar ciertos puntos importantes relativos a la experiencia de usuario, y al tiempo de respuesta para la resolución de casos realistas del problema. La interfaz de usuario de la aplicación móvil desarrollada se presenta en la Figura 8.

Un punto importante es que la aplicación móvil solamente funciona como intermediario entre el usuario móvil y el Servidor Central. El cálculo de la asignación de pasajeros y recorridos de los taxis, al igual que en la versión web, se realiza en el Servidor Central. La responsabilidad de la aplicación móvil es mostrar un mapa y almacenar los puntos. Luego de enviarlos al servidor, la aplicación queda a la espera de los resultados del AE para mostrarlos en la pantalla del usuario.

VIII. Conclusiones

Este trabajo ha presentado el diseño e implementación de un algoritmo evolutivo para la resolución del problema de planificación en línea de viajes compartidos en taxis desde un origen a múltiples destinos.

El algoritmo evolutivo propuesto conforma una precisa y eficiente solución para el problema abordado. El análisis experimental realizado usando un conjunto de instancias del problema construidas en base a datos reales tomados de GPS de taxis de la ciudad de Beijing, mostró que los resultados del algoritmo evolutivo permiten mejorar significativamente—hasta un **41.7** % en el mejor caso y **36.4** % en el caso promedio—los resultados sobre los calculados empleando un algoritmo ávido que simula el comportamiento intuitivo de un grupo de usuarios humanos al enfrentarse al problema. Los resultados se obtienen en tiempos de ejecución reducidos, permitiendo su aplicación para la resolución en línea del problema.

Adicionalmente, se presentó el desarrollo de un sistema que incluye una aplicación web y una aplicación móvil para dispositivos iOS, que permiten a los usuarios finales resolver instancias reales del problema de una forma simple y amigable. Dado que las instancias utilizadas en el análisis experimental del método propuesto tienen dimensiones mayores a las que usualmente se proponen resolver en la práctica por usuarios reales a través de una aplicación web o móvil, es de esperar que el algoritmo evolutivo mantenga, o incluso supere, el muy buen desempeño obtenido para las instancias abordadas en este trabajo.

Las principales lineas de trabajo futuro se orientan a mejorar el desempeño del algoritmo evolutivo propuesto, para mejorar su aplicabilidad a casos realistas de alta demanda. En este sentido, el uso de modelos paralelos que permitan obtener soluciones de calidad reduciendo el tiempo de ejecución, surge como una de las principales alternativas a explorar en esta línea de trabajo. Adicionalmente, la posibilidad de estudiar modificaciones del problema considerado puede resultar de gran interés. Entre las variantes en las que se está trabajando actualmente se puede mencionar las que incorporan restricciones temporales al problema, preferencias personales de los pasajeros, y también las que plantean incorporar datos realistas del tráfico de las ciudades. Por último, otras metaheurísticas podrían ser estudiadas con el objetivo de analizar su adaptabilidad a las propiedades del problema, para potencialmente encontrar mejores soluciones.

En la página web del proyecto http://www.fing.edu.uy/inco/grupos/cecal/hpc/AG-Taxi se encuentran publicados los siguientes recursos: i) ejecutable del algoritmo evolutivo y del algoritmo ávido con instrucciones; ii) generador de instancias del problema; iii) instancias del problema y resultados numéricos detallados de los experimentos de calibración y evaluación del algoritmo evolutivo propuesto; y iv) link a la interfaz web y a la aplicación móvil.

Agradecimientos

Las actividades de investigación reportadas en este artículo han sido parcialmente financiadas por ANII y PEDECIBA, Uruguay.

Referencias

[1] N. Fellows and D. Pitfield, "An economic and operational evaluation of urban car-sharing," *Transportation Research Part D: Transport and Environment*, vol. 5, no. 1, pp. 1–10, 2000.

[2] E. Ferrari, R. Manzini, A. Pareschi, A. Persona, and A. Regattieri, "The car pooling problem: Heuristic algorithms based on savings functions," *Journal of Advanced Transportation*, vol. 37, pp. 243–272, 2003.

[3] R. Katzev, "Car sharing: A new approach to urban transportation problems," *Analyses of Social Issues and Public Policy*, vol. 3, no. 1, pp. 65–86, 2003.

[4] A. Letchford, R. Eglese, and J. Lysgaard, "Multistars, partial multistars and the capacitated vehicle routing problem," *Mathematical Programming*, vol. 94, no. 1, pp. 21–40, 2002.

[5] S. Nesmachnow, "Metaheuristics as soft computing techniques for efficient optimization," in *Encyclopedia of Information Science and Technology*, M. Khosrow-Pour, Ed. IGI Global, 2014, pp. 1–10.

[6] "How to Split a Shared Cab Ride? Very Carefully, Say Economists," The Wall Street Journal, 8 de diciembre, 2005, [Online] http://online.wsj.com/news/articles/SB113279169439805647; accessed 07-04-2014.

[7] C.-L. Xin and W.-M. Ma, "Scheduling for on-line taxi problem on a real line and competitive algorithms," in *Proceedings of the International Conference on Machine Learning and Cybernetics*, vol. 5, Aug 2004, pp. 3078–3083.

[8] E. Koutsoupias, "The k-server problem," *Computer Science Review*, vol. 3, no. 2, pp. 105–118, 2009.

[9] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 2013, pp. 410–421.

[10] C.-C. Tao and C.-Y. Chen, "Heuristic algorithms for the dynamic taxipooling problem based on intelligent transportation system technologies," in *4th International Conference on Fuzzy Systems and Knowledge Discovery*, vol. 3, Aug 2007, pp. 590–595.

[11] T. Bäck, D. Fogel, and Z. Michalewicz, Eds., *Handbook of evolutionary computation*. Oxford University Press, 1997.

[12] "The Malva Project: A framework for comutational intelligence in C++," [Online] https://github.com/themalvaproject, accessed 04-04-2014.

[13] E. Alba, G. Luque, J. Garcia-Nieto, G. Ordonez, and G. Leguizamon, "Mallba; a software library to design efficient optimisation algorithms," *International Journal of Innovative Computing Applications*, vol. 1, no. 1, pp. 74–85, 2007.

[14] C. M. Schweik, M. T. Fernandez, M. P. Hamel, P. Kashwan, Q. Lewis, and A. Stepanov, "Reflections of an online geographic information systems course based on open source software," *Social Sciences Computing Review*, vol. 27, no. 1, pp. 118–129, Feb. 2009.

[15] "Google Earth," [Online] http://www.google.com/intl/en/earth/, accessed 19-04-2014.

[16] "TaxiFareFinder API (beta)," [Online] http://www.taxifarefinder.com/api.php, accessed 19-04-2014.

[17] E. Cantu-Paz and D. E. Goldberg, "Are multiple runs of genetic algorithms better than one?" in *Proceedings of the Genetic and Evolutionary Computation Conference*. Springer Verlag, 2003, pp. 801–812.

[18] S. Shapiro and M. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

[19] N. Razali and Y. Wah, "Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests," *Statistics and Computing*, vol. 2, no. 3, pp. 117–119, 2011.

[20] H. Mann and D. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.

[21] N. Middleton and R. Schneeman, *Heroku: Up and Running*. O'Reilly Media, Inc., 2013.

[22] B. Tate and C. Hibbs, *Ruby on Rails: Up and Running*. O'Reilly Media, Inc., 2006.

[23] D. Flanagan and Y. Matsumoto, *The Ruby Programming Language*. O'Reilly, 2008.

[24] J. Wargo, *PhoneGap Essentials: Building Cross-Platform Mobile Apps*. Addison-Wesley Professional, 2012.